RANDOM MEMORY

Documentació

Sergio Martos Forniés
16 d'Octubre de 2015
Treball Final de Grau – Videojocs Educatius
Universitat Oberta de Catalunya

Índex

1. Actualitzacions

Canvis amb respecte versions anteriors del document.

- **Revisió 1**
  - Correccions menors en tot el document.
- **Revisió 2**
  - Correccions menors en tot el document.
  - Substitució de captures anteriors per actuals.
  - Extensió de les descripcions.
  - Explicació del desenvolupament en Unity3D
  - Explicació dels scripts que composen l'aplicació

2. Introducció

Aquest és el document de *Random Memory*. És el videojoc per dispositius mòvils amb sistema operatiu Android que exercita la memòria de l'usuari, utilitzant Unity3D com a plataforma de desenvolupament de videojocs. Aquest escrit té com a objectiu principal detallar els elements que ha d'incloure *Random Memory* i servir de carta de presentació.

2.1. Concepte del joc

Random Memory és un videojoc en el que es mostren una sèrie de cartes dins d'un escenari perquè endivinem totes les parelles idèntiques sota la presió del temps contrarrellotge abans que aquest finalitzi. Aquestes parelles poden ser formades per imatges o sons, segons el mode de joc triat per l'usuari.

2.2. Característiques principals

El joc es basa en els següents fonaments:

- **Plantejament senzill**: La complexitat és mínima, no té història ni missions complicades. Però, si conté un sistema de puntuació i classificació perquè el jugador senti que té un objectiu.

- **Memòria:** Endivinar totes les parelles de cartes idèntiques ha de ser quasi impossible si es comença a escollir cartes aleatòriament. L'elecció de les cartes per afavorir la memòria d'aquestes serà necessari.

- **Dinamisme:** *Random Memory* ha de ser dinàmica i provocar una sensació de tensió creixent en el jugador a mesura que s'esgota el temps.

- **Ampliació:** *Random Memory* ha de ser ampliable amb nous escenaris i nous modes de joc de forma senzilla. El funcionament del joc serà lo més independent possible del contingut. D'aquesta manera, es podrà afegir noves actualitzacions en el menor temps possible.

2.3. Gènere

Random Memory és de gènere **arcade.** Els videojocs d'aquest tipus es caracteritzen per la seva jugabilitat simple, repetitiva i acció ràpida. També compten amb uns gràfics i/o argument més simples. *Random Memory* conté una jugabilitat senzilla on l'acció principal és donar la volta a les cartes per mostrar el seu valor. Els esdeveniments que surgeixen segons els modes de joc tampoc tenen una gran complexitat.

2.4. Tecnologia

Random Memory és una aplicació destinada a dispositius mòvils amb sistema operatiu Android, tant per *tablets* com per *smartphones*.

Per desenvolupar el projecte utilitzarem l'eina de desenvolupament *Unity3D*. Aquesta plataforma és una eina de desenvolupament d'aplicacions gratuïta, flexible i robusta, que pretén que els desenvolupadors aconsegueixin crear jocs o experiències 2D o 3D multiplataforma. Per tant, part del desenvolupament del videojoc es farà desde l'entorn gràfic.

Unity3D també ens ofereix un entorn de desenvolupament de codi anomenat *MonoDevelop*, que suporta els llenguatges de programació C# i Javascript. En *Random Memory* es desenvoluparà tots els scripts en C#.

Finalment, integrarem *Unity3D* amb *Google Play Games Services* mitjançant complements, per aportar al videojoc funcions socials com assoliments i classificacions.

2.5. Propòsit i públic objectiu

El principal objectiu de *Random Memory* és oferir als usuaris de la plataforma Google Play un producte jugable i divertit, on els més hàbils poden competir en les classificacions oferides per les funcions socials de Google.

Random Memory està adaptat a jugadors d'un rang ampli d'edats amb temps limitat per dedicar a l'oci electrònic. Degut això, el sistema del joc es basa en partides de curta duració i sense cap tipus de relació històrica entre elles, fet que permet poder jugar de forma esporàdica. No obstant, per als jugadors que vulguin jugar més estona se li presenten escenaris i cartes totalment aleatories i diferents modes de joc per tal de que l'experiència no sigui monòtona.

2.6. Jugabilitat

Cada nivell de *Random Memory* mostra un escenari aleatori juntament amb el temps i cartes corresponents. Hem d'aconseguir trobar totes les parelles de cartes idèntiques abans de que s'esgoti el temps. Per això, ens basem en els següents elements:

- **Valor de les cartes:** El jugador podrà escollir únicament dos cartes, quatre en el mode de doble parella, les quals mostraran el seu valor en un temps molt limitat. En cas d'encertar, les cartes romandran girades mostrant el seu valor fins el final de la partida. En cas contrari, es tornaran a donar la volta ocultant el seu contingut.

- **Efectes visuals:** Segons els modes de joc, el jugador rebrà inconvenients segons la seva jugada. Entre els inconvenients poden aparèixer cartes dinàmiques, les quals canvien de posició segons les errades de l'usuari.

- **Bonus:** Entre els avantatges es poden presentar el multiplicador de puntuació. Segons els encerts seguits que aconsegueix el jugador la puntuació es va multiplicant.

2.7. Estil visual

Random Memory tindrà un estil senzill, no massa detallista i aleatori. És a dir, cada escenari no tindrà res a veure amb l'anterior cada cop que el jugador comenci una nova partida. Per tant, existirà més d'un estil visual com poden ser: retro, dibuixos animats, escenaris realistes, etc. Amb aquest fet es pretén que el joc sigui variable i entretingut.

2.8. Abast

L'objectiu principal es desenvolupar un sistema de joc el qual podem introduïr nous continguts sense dificultat. En primera instància es desenvoluparà un seguit de modes de joc que seran ampliats en un futur.

3. Mecàniques de joc

En aquesta secció entrarem més en detall en les mecàniques de *Random Memory.* S'exposaran tots els pilars fonamentals que sostenen la seva jugabilitat així com les accions que podrà duu a terme l'usuari dins d'una partida.

3.1. Jugabilitat

La jugabilitat està formada per:

- **Escenaris:** Cada nivell de *Random Memory* és un escenari o paisatge aleatori. Aquests escenaris actuaran com a tauler i donaran l'efecte visual de soportar les cartes.

- **Cartes:** L'usuari podrà escollir cartes inmediatament i aquestes mostraran el seu contingut. El sistema comprovarà si son iguals en un temps limitat on l'usuari no podrà escollir cap carta més. Un cop finalitzada la comprovació, les cartes romandran girades o es donaran la volta segons si el jugador ha encertat o no, i podrà tornar a escollir cartes.

- **Temps a contrarellotge:** El temps és l'enemic principal del jugador. Quan comença la partida el temps comença a comptar enrere fins arribar a zero, donant per perduda la partida.

- **Intensitat:** Segons les jugades realitzades i modes de joc, la dificultat de *Random Memory* es veu incrementada amb els efectes visuals esmentats anteriorment.

- **Planificació de l'elecció de les cartes:** El jugador haurà d'escollir les cartes adequades per tal d'afavorir la seva pròpia memòria i el transcurs de la partida.

3.2. Flux del joc

En aquesta secció es detallarà els transcurs d'una partida típica a *Random Memory.* Es comentaran els passos que el jugador ha de seguir desde que inicia l'aplicació fins a completar una partida. Per tant, aquí mostrem les mecàniques i més endevant es definirà el contingut de les interfícies.

El jugador inicia *Random Memory* i se li presenta el menú principal. El menú principal consta de cinc opcions bàsiques: jugar, classificació, assoliments, ajuda i sortir. Donades aquestes cinc opcions:

- Si el jugador seleccionar l'opció per jugar, se li presenta un altre menú amb

els diferents modes de joc.

- Per poder veure la classificació i els assoliments, cal que l'usuari iniciï la sessió en Google Play. Per tant, aquestes es mostraran deshabilitades, i al seleccionar qualsevol de les dos demanarà el permís al jugador per iniciar la sessió. Un cop iniciada, al tornar a seleccionar les opcions es mostraran la classificació i assoliments respectivament desde la plataforma externa de Google Play.

- Si el jugador prem l'opció d'ajuda, se li mostra una pantalla amb una descripció bàsica de l'objectiu del joc.

- Si el jugador toca l'opció de sortir es tanca l'aplicació.

Un cop en el menú de la selecció de modes de joc el jugador trïa un mode de joc. Seguidament, el sistema carrega un escenari aleatori amb el temps aturat i les cartes corresponents voltejades ocultant el seu contingut.

Un cop comença la partida, el temps comença a comptar enrere i l'usuari ha de seleccionar les cartes on les parelles siguin idèntiques. L'usuari selecciona una carta, aquesta dona la volta per mostrar el seu valor, llavors pot seleccionar la segona carta del torn, la qual també mostra el contingut ocult. El sistema comprova el valor de les cartes per determinar si són iguals. Mentrestant, en un temps limitat totes les cartes pasen a ser no seleccionables. Per tant, l'usuari no pot seleccionar cap carta més fins que no finalitzi la comprovació.

Si l'usuari encerta la parella del mateix valor, aquesta romandrà girada mostrant el seu valor fins la resta de partida i pasaran a ser no seleccionables. En cas contrari, les cartes es tornen a girar. Un cop finalitzada la comprovació, les cartes no encertades i la resta passen a ser seleccionables de nou.

Si el jugador aconsegueix endevinar totes les parelles de cartes abans de que el temps finalitzi s'aturarà la partida mostrant un missatge de victoria juntament amb la puntuació i rècords, on aquest últim s'actualitzarà si és necessari. En cas contrari, es mostrarà una pantalla similar però amb el missatge de derrota i sense puntuacions.

Tant si guanya com si perd, desde la pantalla que es mostra el jugador té dos opcions, tornar a jugar o tornar al menú principal. Més endevant es detallaran totes les possibilitats del flux entre pantalles.

3.3. Moviment i físiques

Random Memory es desenvolupa sobre un entorn 2D simple en el que l'escenari no té cap tipus d'efecte físic en les cartes ni viceversa. Per tant, les cartes només es veuen afectades físicament a partir de les animacions corresponents segons les circumstàncies de la partida.

3.3.1. Interacció entre elements

En *Random Memory* l'única interacció que es presenta és l'existent entre l'usuari i les cartes. L'usuari toca la pantalla del dispositiu, la carta en la posició concreta detecta la col·lisió i reacciona segons els transcurs de la partida.

3.3.2. Controls

Els controls en *Random Memory* es basen únicament en la pantalla tàctil del dispositiu mòvil. Dins de la partida no s'ofereix cap tipus de botó, el contacte és directe entre la pantalla i la carta.

4. Interfície

En aquesta secció s'especificarà amb detall cada una de les pantalles que formen *Random Memory.* A més a més, s'indicaran les transicions entre elles així com la funció de cada element de l'interfície gràfica.

4.1. Diagrama de flux

El següent diagrama d'estats mostra les pantalles presents en l'aplicació i les transicions entre elles. En els següents apartats detallarem l'activitat de les interfícies individualment.

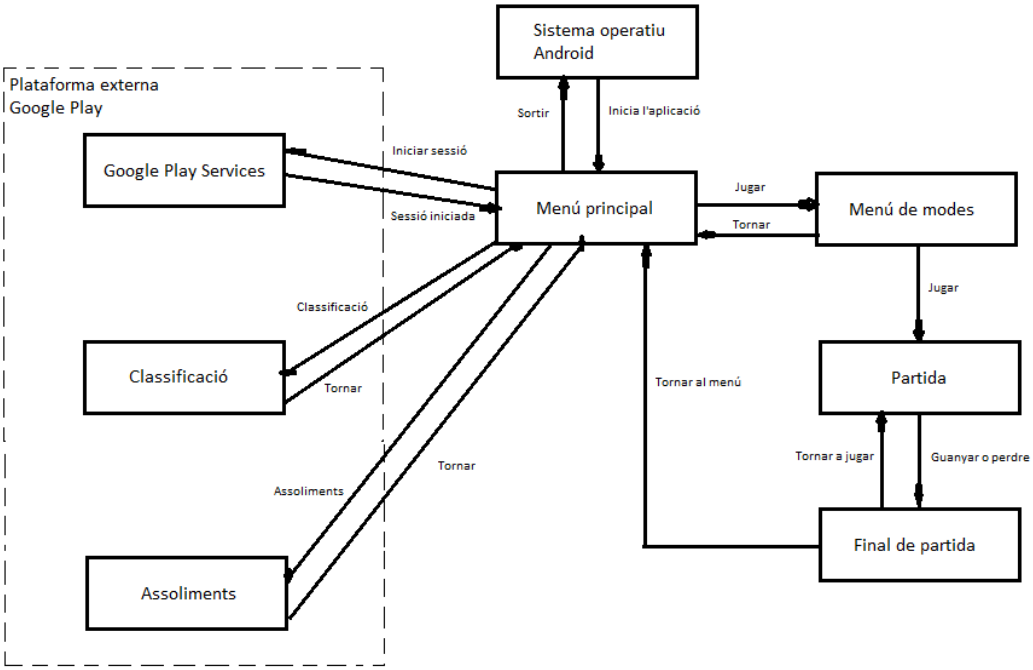


Figura 1: Diagrama de flux de les pantalles del joc.

4.2. Menú principal

A continuació el disseny de la pantalla de *Menú Principal*:

Figura 2: Disseny del menú principal.

Llista i descripció del seus components:

- **Botó jugar:** Al tocar-ho porta a la pantalla *Selecció de modes*.
- **Botó classificació:** Al tocar-ho porta a la plataforma externa de Google play, tant per iniciar sessió com per veure la classificació.
- **Botó assoliments:** Al tocar-ho porta a la plataforma externa de Google play, tant per iniciar sessió com per veure els assoliments aconseguits.
- **Botó ajuda:** Al tocar-ho porta a la pantalla *Ajuda*.
- **Botó sortir:** Al tocar-ho tanca l'aplicació i retorna al sistema operatiu.

4.3. Selecció de modes

A continuació el disseny de la pantalla de *Selecció de modes*:

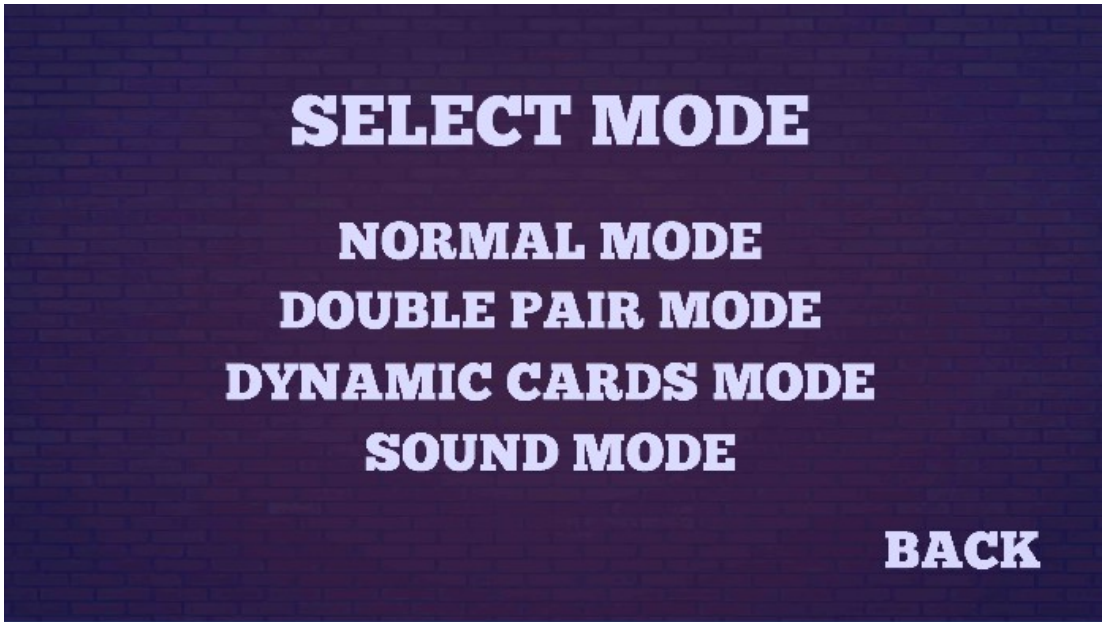


Figura 3: Disseny de la selecció de modes.

Llista i descripció del seus components:

- **Llistat de modes:** Al tocar un mode porta a la pantalla de *Partida* específica al mode de joc corresponent.
- **Botó mode normal:** Al tocar aquest mode porta a la pantalla de *Partida* on es crearan les característiques principals del mode normal.
- **Botó mode doble parella:** Al tocar aquest mode porta a la pantalla de *Partida* on es crearan les característiques principals del mode de doble parella.
- **Botó mode dinàmic:** Al tocar aquest mode porta a la pantalla de *Partida* on es crearan les característiques principals del mode dinàmic.
- **Botó mode so:** Al tocar aquest mode porta a la pantalla de *Partida* on es crearan les característiques principals del mode de so.
- **Botó torna:** Al tocar-ho retorna al *Menú principal*.

4.4. Partida

A continuació el disseny de la pantalla de *Partida*:

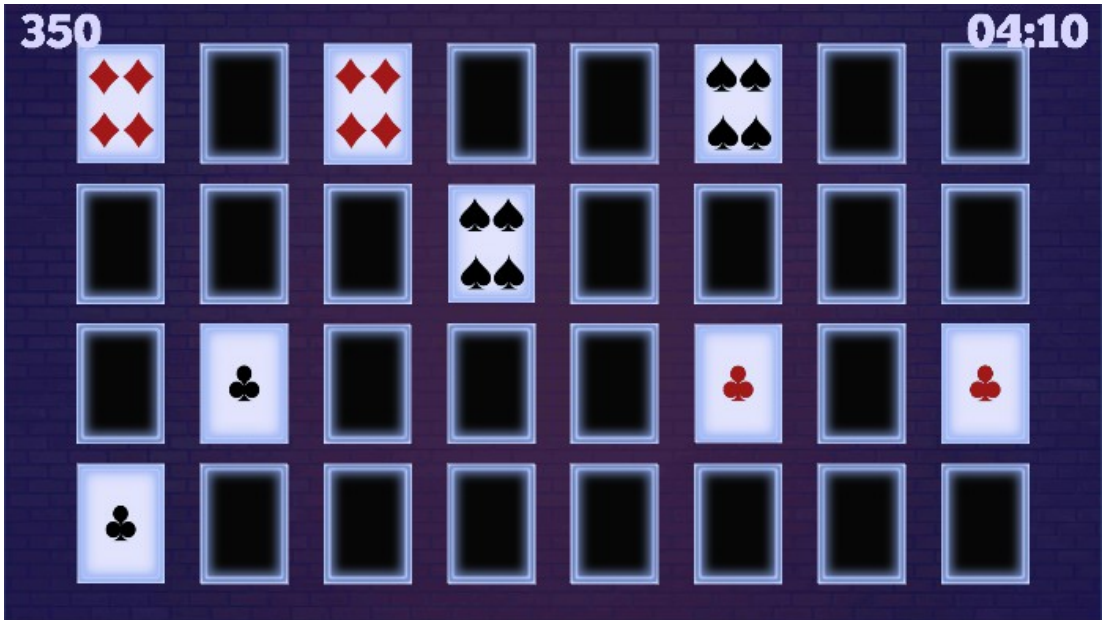


Figura 4: Disseny de la partida.

Llista i descripció del seus components:

- **Escenari:** Tauler del joc.
- **Contrarrellotge:** Indica el temps restant per finalitzar la partida. Quan arriba a zero mostra el *Final de partida.*
- **Puntuació:** Indica la puntuació actual que l'usuari ha aconseguit. Segons els encerts seguits que el jugador realitza, es mostra el multiplicador per aconseguir puntuació extra.
- **Cartes:** Al tocar-les, sempre que siguin seleccionables realitzaran l'animació de girar 180 graus fins mostrar el seu contingut.

4.5. Final de partida

A continuació el disseny de la pantalla de *Final de partida*:

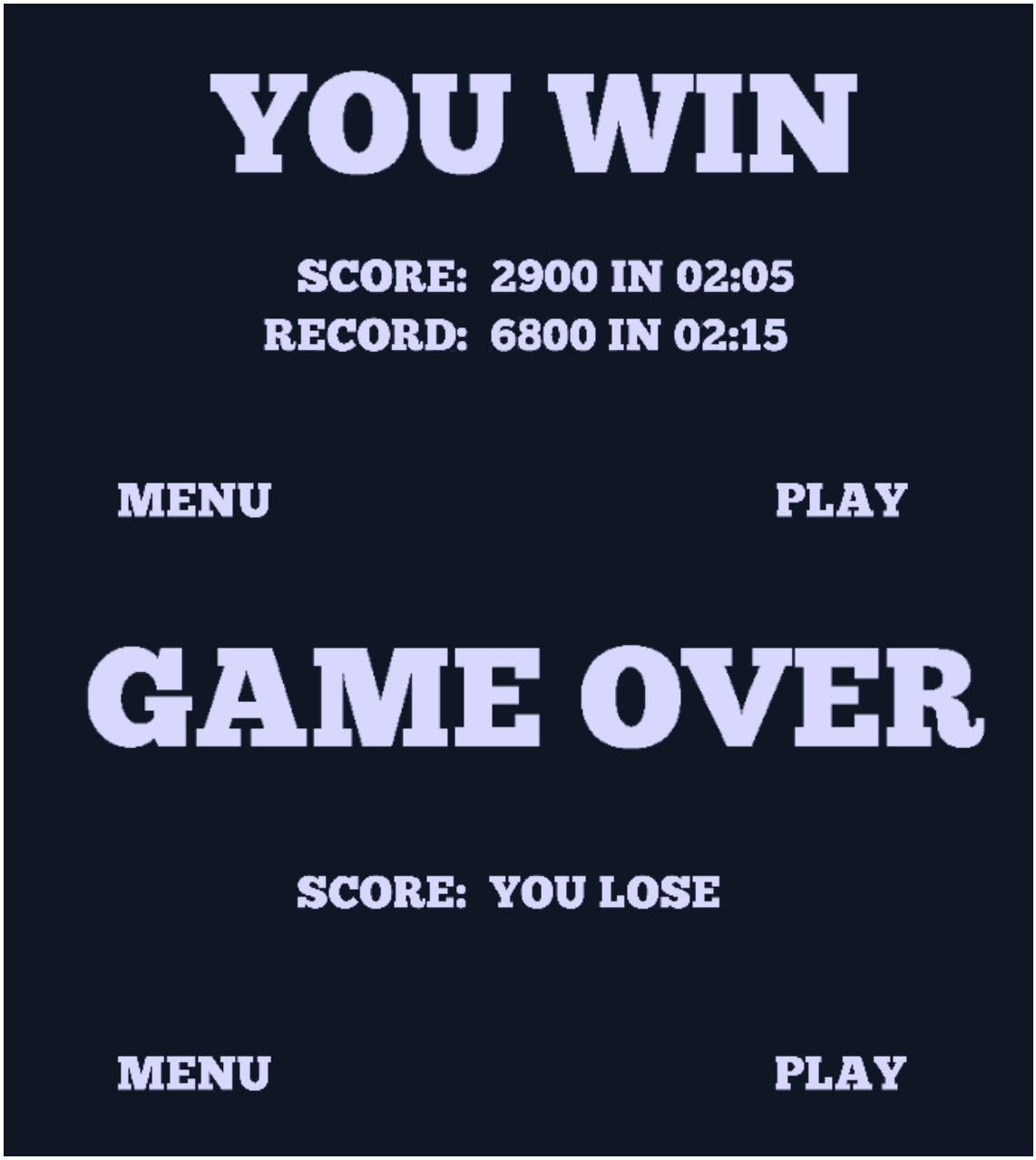


Figura 5: Disseny del final de partida.

Llista i descripció del seus components, segons la pantalla mostrada:

- **Missatge:** Mostra el missatge de final de partida "You win" o "Game over".

- **Puntuació:** Indica la puntuació aconseguida en la partida.
- **Rècord:** Indica la puntuació màxima aconseguida pel jugador.
- **Botó menú:** Al tocar-ho retorna a la pantalla de *Menú principal*.
- **Botó jugar:** Al tocar-ho retorna a la pantalla de *Partida*.

4.6. Ajuda

A continuació el disseny de la pantalla d'*Ajuda*:

Figura 6: Disseny de l'ajuda.

Llista i descripció del seus components:

- **Descripció:** Mostra una breu descripció de l'objectiu principal del joc.
- **Botó torna:** Al tocar-ho retorna al *Menú principal*.

4.7. Classificació

A continuació el disseny, ofert per la plataforma externa Google Play Games, de la pantalla de *Classificació*:

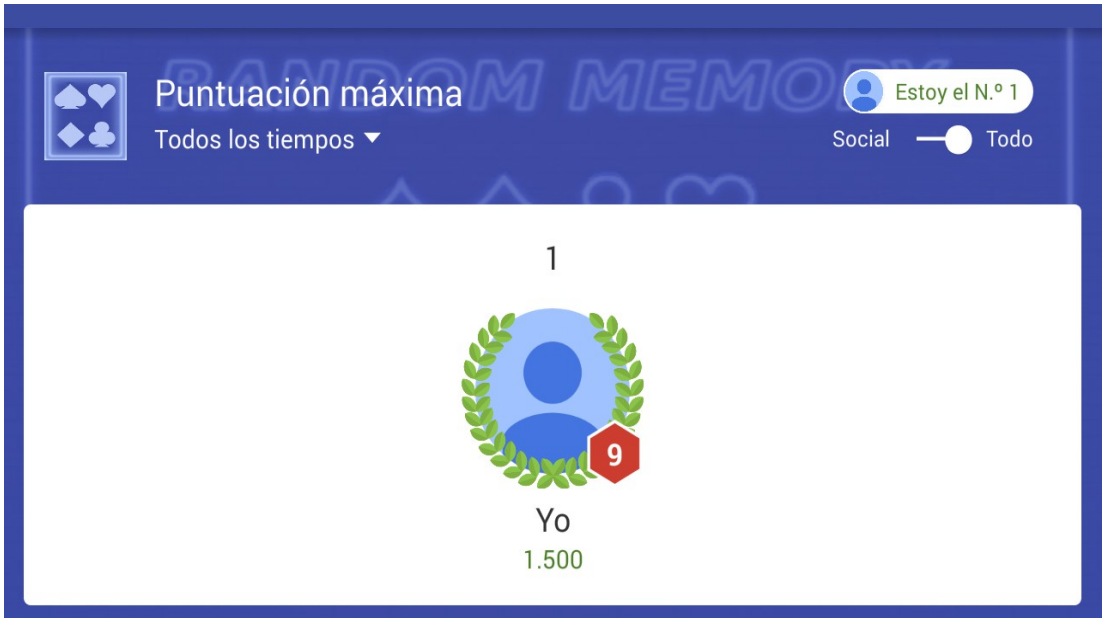


Figura 7: Disseny de la classificació.

Llista i descripció del seus components:

- **Percentatge de la posició:** Indica la posició/percentatge del jugador en la classificació.
- **Eines de cerca:** Opcions per l'usuari per facilitar la cerca desitjada.
- **Els 3 millors jugadors:** Indica els 3 jugadors amb les millors puntuacions. Mostra la imatge de perfil, el nom, nivell i puntuació de l'usuari.
- **Llistat de la resta de jugadors:** Llistat, ordenat per puntuació, d'usuaris on es mostra la mateixa informació que els tres millors jugadors.

4.8. Assoliments

A continuació el disseny, ofert per la plataforma externa Google Play Games, de la pantalla d'*Assoliments*:

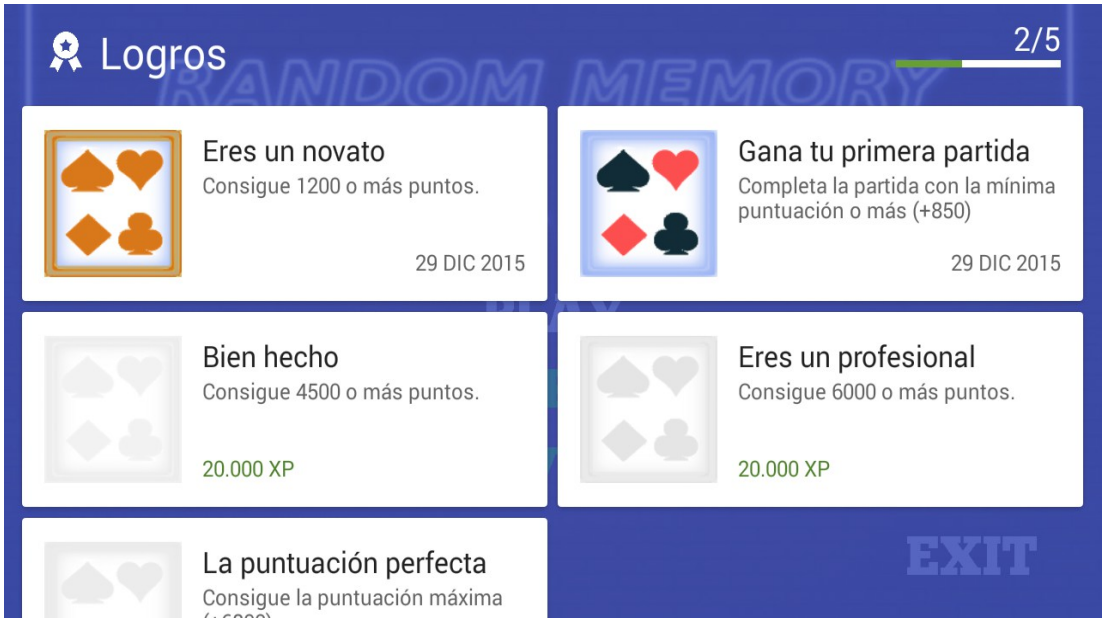


Figura 8: Disseny dels assoliments.

Llista i descripció del seus components:

- **Barra de percentatge d'assoliments:** Indica el nombre d'assoliments aconseguits i total, juntament amb el percertatge de progrés.

- **Assoliments:** Llistat amb els assoliments aconseguits i per desbloquejar. Mostra la imatge, nom, descripció i l'experiència adquirida en el perfil de Google Play Games.

5. Art

Random Memory no ha de tenir cap estàndar a l'hora de definir un estil gràfic. Com el seu propi nom indica, els escenaris, cartes i temporitzador seran aleatoris. No obstant, és necessari que els colors siguin tan agradables com còmodes a la vista.

Totes les imatges hauran d'estar en format *.png* a més a més del format propi del programa amb el que es creen, per posibles modificacions.

Les imatges importades a l'entorn gràfic són:

- **Pantalla principal**: Ambientat en l'oci nocturn. Pared fosca amb llum central, on el logo està realitzat amb llums de neó blava.
- **Escenari**: Aleatori.
- **Part de darrera de la carta**: Aleatori, però adaptada a l'escenari.
- **Part de davant de la carta**: Aleatori, però adaptada a l'escenari.

Un cop feta la introducció, passem a veure tots els dissenys dels escenaris del joc:

Escenari 1

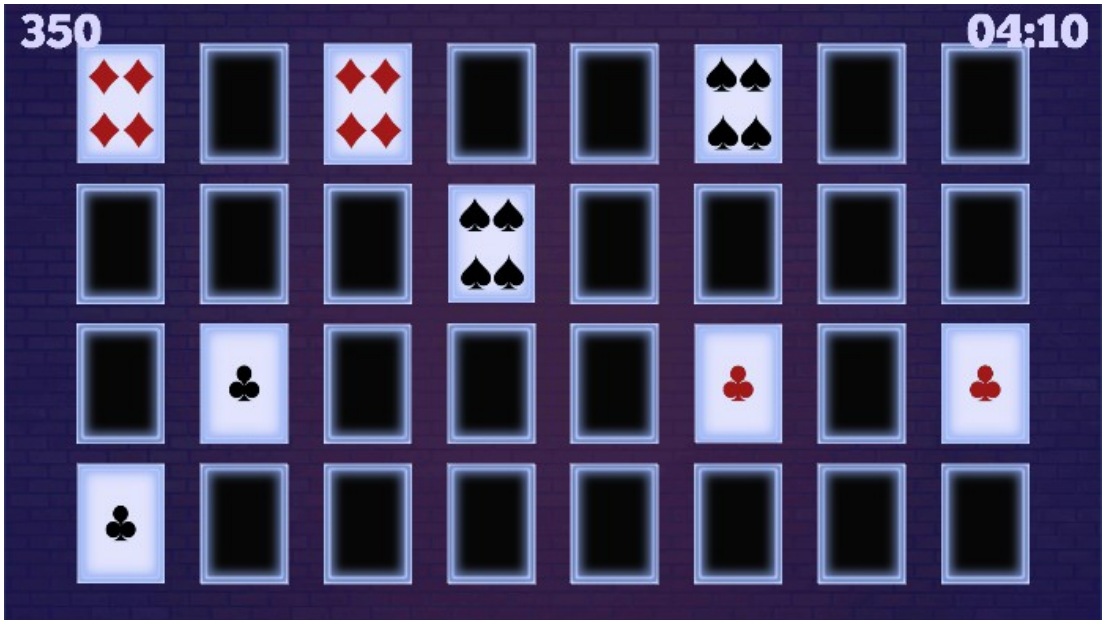


Figura 9: Escenari 1

Aquest és l'escenari principal, ja que concorda amb el disseny del menú principal.

El fons és una pared de maons amb colors foscos simulant l'ambient nocturn. Les cartes són de color negre per darrera, envoltades d'una llum de neó. Per davant tenen un color de fons més clar i els valors són els de la baralla del póker.

Escenari 2

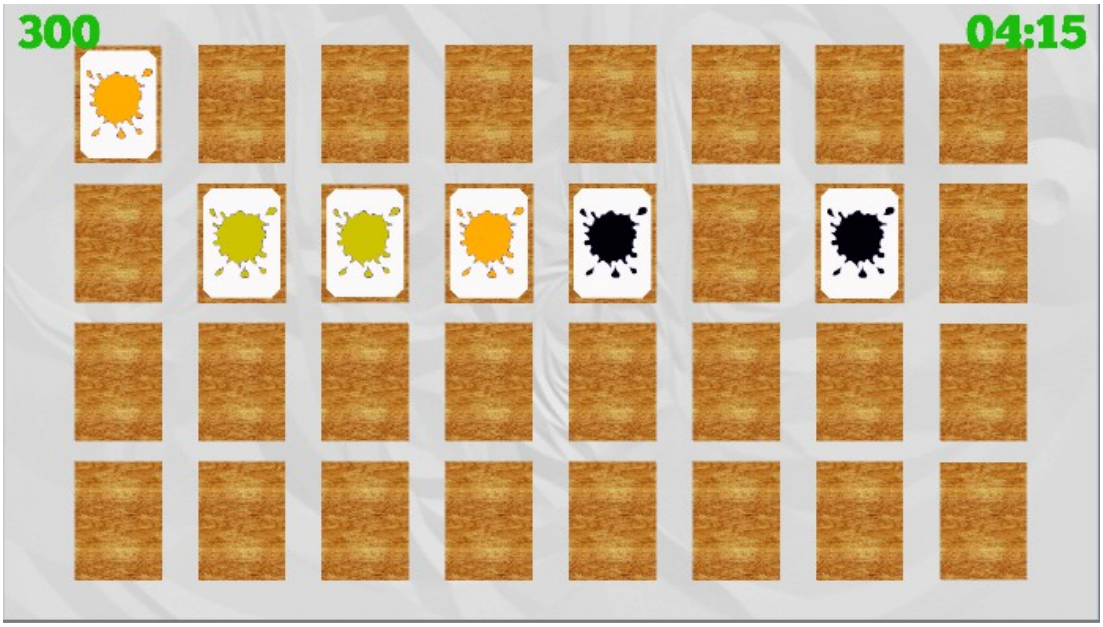


Figura 10: Escenari 2

Aquest és el segon escenari, ambientat en l'art i en la pintura. El fons és un paper arrugat que simula l'ambient de l'art. Les cartes són de color marró per darrera, fent que sembli fusta. Per davant mantenen el voltant de la fusta i per dins són de color blanc amb la taca de pintura del color corresponent, fent que la carta simuli un petit taulell de fusta amb un paper a sobre.

Escenari 3

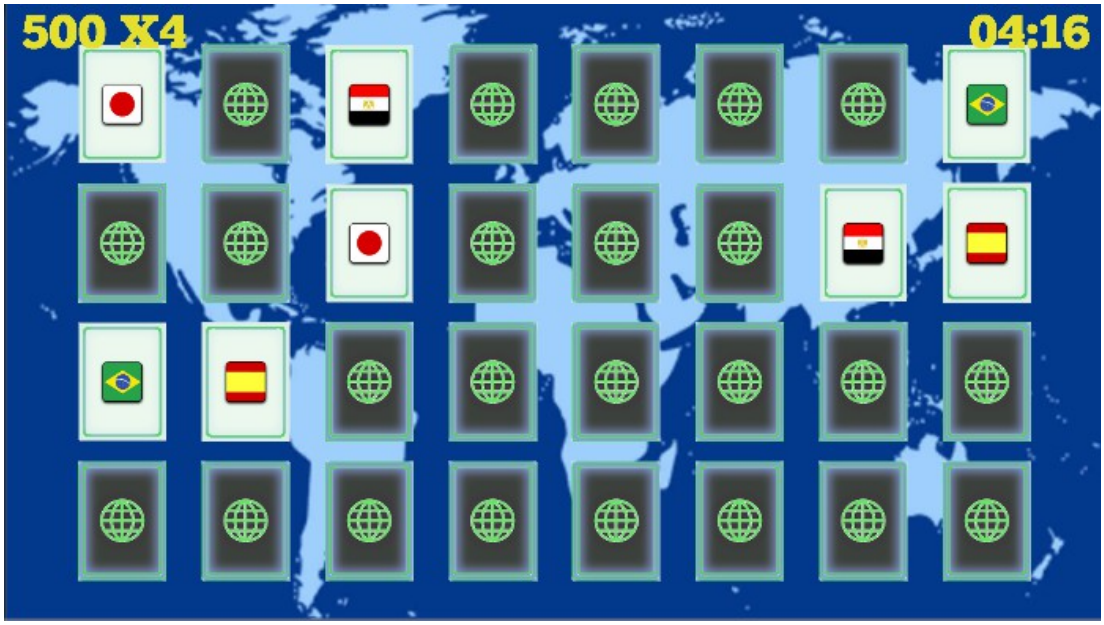


Figura 11: Escenari 3

Aquest és el tercer escenari, ambientat al mapa del planeta Terra. El fons conté la figura del mapa mundial representat per colors blaus, donant així un ambient menys colorit per poder combinar amb els altres elements de la pantalla. Les cartes són de color negre per darrera, envoltades d'una llum de neó verd i en el centre tenen el símbol global del mateix color. Per davant tenen un color de fons més clar, envolat per un rectangle verd i els valors són les banderes nacionals de diferents països del món de forma quadrada.

Escenari 4

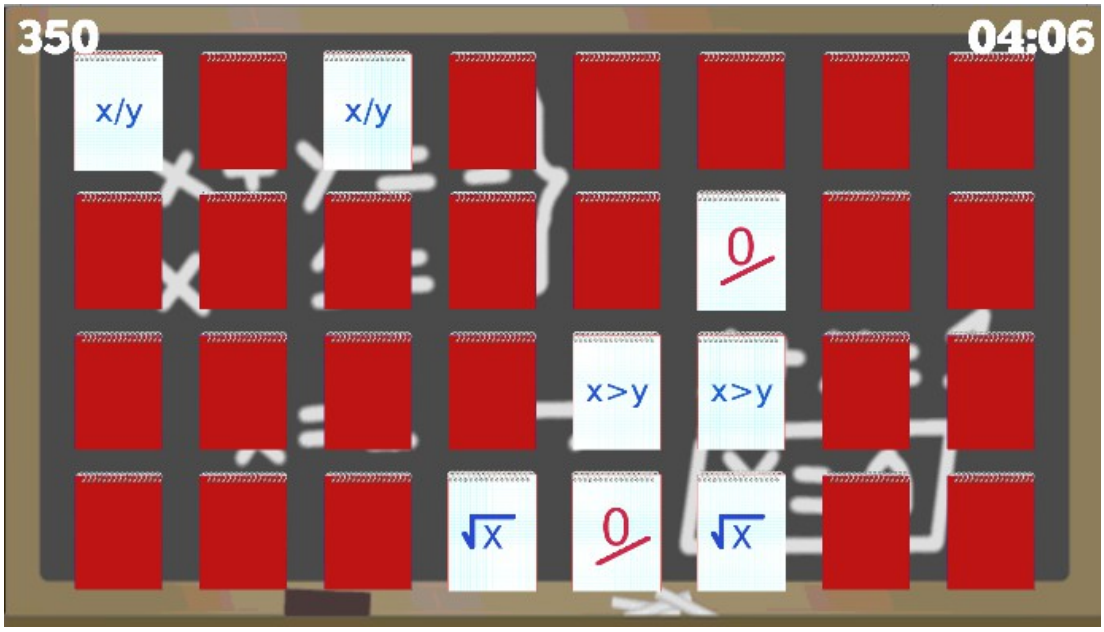


Figura 12: Escenari 4

Aquest és el quart escenari, ambientat en una aula escolar. El fons és una pissarra guixada amb un sistema d'equacions simple i resolt. Les cartes són de color vermell per darrera amb unes anelles a la part superior. Per davant tenen un color de fons clar amb linies blaves, mantenint les anelles a la part superior, i els valors són operacions matemàtiques, fent que la carta simuli una petita llibreta.

Escenari especial

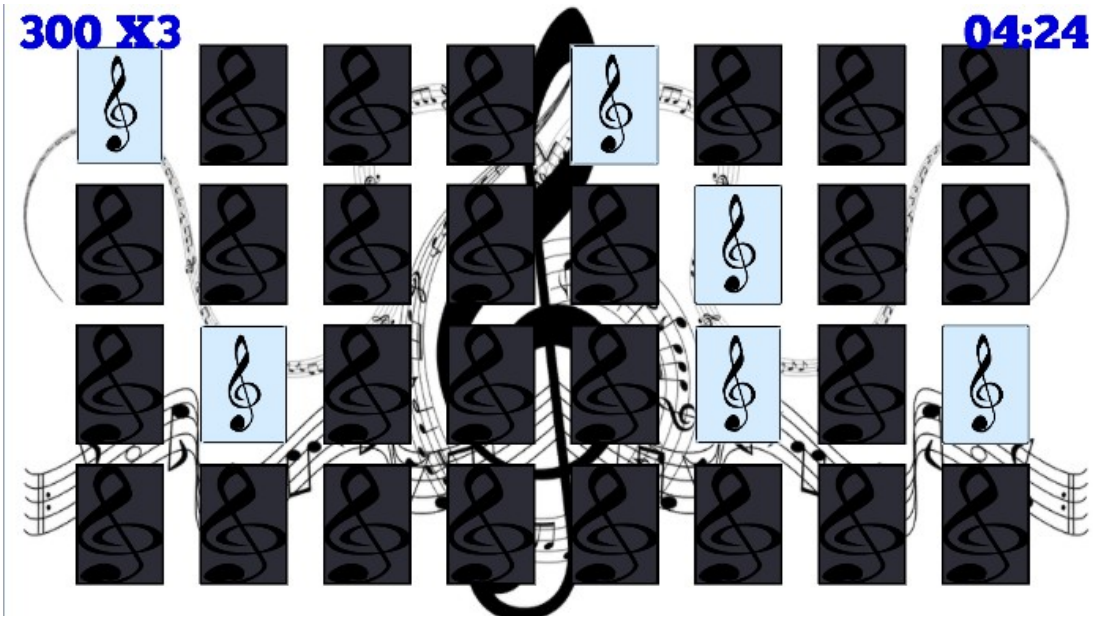


Figura 13: Escenari especial

Aquest és un escenari especial que només es mostra al mode de so. El fons és blanc amb partitures colocades de manera que formen una figura. Les cartes són de color negre per darrera amb una clau de sol estirada. Per davant tenen un color de fons clar amb una clau de sol més comprimida, fent que l'escenari estigui ambientat en el món de la música.

5.1 Àudio

Com amb les imatges, és necessari guardar l'arxiu d'àudio en el format amb el programari que es produeixi. Tant la música com els efectes de so estaran en format *.wav.*

Música:

- Menú principal.
- Selecció de modes.
- Partida.

Efectes:

- Seleccionar botó.
- Seleccionar carta.
- Finalitzar la partida.

Un cop feta la introducció, passem a veure tots els sons del joc:

Pantalla de menú, selecció de modes i pantalla d'ajuda

En aquestes pantalles reproduïm l'arxiu *Intro.wav* com a música de fons dels menús, a mode de repetició. Aquest tema està pensat per ambientar els menús de l'aplicació amb una melodia tranquila, acollidora i fàcil de recordar, sense crear tensió a l'usuari.

Pantalla de partida

En aquest cas reproduïm l'arxiu *inGame.wav* com a música de fons de la partida. Aquest tema està pensat per ambientar la partida amb una melodia tranquila i més greu que l'anterior, de manera que l'usuari es pugui concentrar més en les cartes.

Botons

En tots els botons de l'aplicació reproduïm l'arxiu *pressButton.wav* quan l'usuari toca l'opció. Es tracta d'un so molt breu amb temàtica retro.

Final de partida

Segons si l'usuari guanya o perd, es reprodueix un so o un altre. Per la victoria reproduïm *win.wav*, per la derrota reproduïm ***over-mono.wav*. Els dos arxius són de curta duració però la diferència principal entre ells és que el primer té una melodia alegre, on el so passa de ser més greu a més agut i el segon passa a ser de més agut a més greu.

Cartes

Totes les cartes seleccionables dins d'una partida de qualsevol mode, excepte el mode de so, reprodueixen l'arxiu *card.wav*, que simula el so que fa una carta al donar la volta.

Mode de so

És l'únic mode on no es reprodueix cap música de fons, degut que el jugador ha de poder escoltar bé els sons de les cartes per poder endevinar-les. En canvi, cada parella de cartes reprodueixen un tipus de soroll, siguin animals, armes, notificacions, etc. El llistat d'arxius són els adjunts a les cartes d'aquest mode:

- *alien.wav*
- *coinUp.wav*
- *cow.wav*
- *Dog.wav*
- *duck_quack.wav*
- *guitar.wav*
- *notification.wav*
- *piano.wav*
- *pig.wav*
- *retro.wav*

- *ring.wav*
- *shotgun.wav*
- *sword.wav*
- *trumpet.wav*
- *watch.wav*
- *zombie.wav*

6. Desenvolupament en Unity3D

En aquest apartat ens endinsem en el desenvolupament del projecte. Cal aclarir que no ens centrarem en el funcionament de l'eina sinó en el projecte concretament. No obstant, tot i que no es tracta d'un tutorial sobre Unity3D, si que afegirem explicacions bàsiques i una breu introducció perquè la comprensió del projecte sigui més fàcil.

A mesura que les explicacions es facin, no es tornaran a repetir degut al volum d'informació redundant que suposaria aquest document. Per tant, pot donar la sensació que falten passos a descriure per poder completar el projecte, però això és degut a que no entrem en detall amb l'eina.

6.1. Introducció a l'entorn gràfic

Benvinguts a l'interfície gràfica de Unity3D on s'ha desenvolupat *Random Memory*:

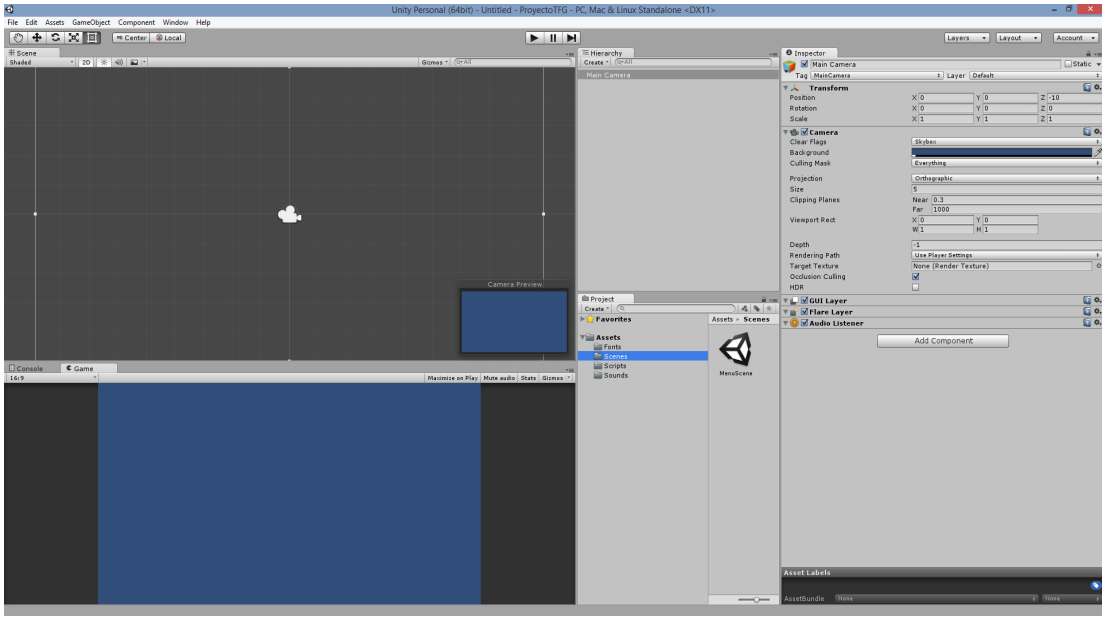


Figura 14: Entorn gràfic d'Unity3D

Explicarem les parts més bàsiques i necessàries per desenvolupar qualsevol projecte en aquest entorn:

- **Pestanya Escena:** És la finestra on es coloquen tots els objectes que es volen visualitzar en el video joc.

- **Pestanya Joc:** És la finestra on es pot visualitzar com es veurà el joc en el dispositiu final. Per reproduïr l'aplicació per veure els resultats tenim els botons sobre la finestra anterior.

- **Pestanya Jerarquia:** És la finestra que conté tots els objectes creats de l'escena actual. Per crear objectes tenim el menú desplegable "create" en la mateixa pestanya.

- **Pestanya Projecte:** És la finestra on es mostren tots els recursos que conté el projecte. Poden ser imatges, escenes, arxius de so, scritps, objectes pre fabricats, plugins, etc.

- **Pestanya Inspector:** És la finestra on, seleccionant un objecte o recurs del projecte, es mostren tots els seus components per poder visualitzar i/o modificar les seves característiques.

Com podem veure, aquesta és una descripció molt breu i a nivell molt bàsic de Unity3D. Aquesta eina presenta una complexitat molt més elevada de la descrita anteriorment i ens ofereix una infinitat de possibilitats a l'hora de crear una aplicació de qualitat.

6.2. Configurant la relació d'aspecte de les càmeres

Què és la relació d'aspecte? La relació d'aspecte d'una imatge o pantalla digital és la proporció entre la seva amplada i la seva alçada, i s'expressa normalment com X : Y. Per exemple, 4:3 és la relació d'aspecte habitual de les televisions juntament amb la panoràmica 16:9, que també es presenta en diferents dispositius com pantalles d'ordinador, dispositius mòbils, etc.

Un dels primers passos importants a l'hora de desenvolupar qualsevol videojoc en Unity és determinar quina serà la visualització que tindrà en el dispositiu final (ordinador, dispositiu mòbil, etc.). Això ens facilitarà, principalment, el posicionament dels elements que contindrà el joc durant el seu desenvolupament. Per tant, s'ha de configurar la relació d'aspecte que tindran les diferents càmeres.

La càmera

La càmera és un dels elements més bàsics i, alhora, importants de qualsevol projecte en Unity. Es tracta d'un element que ens permet mostrar a l'usuari les diferents escenes del videojoc de la mateixa manera que pot representar als espectadors les diferents escenes d'una pel·lícula.

En cada escena hi ha, per defecte, una càmera principal per poder mostrar el contingut a l'usuari, però es poden crear les càmeres necessàries en GameObject -> Camera.

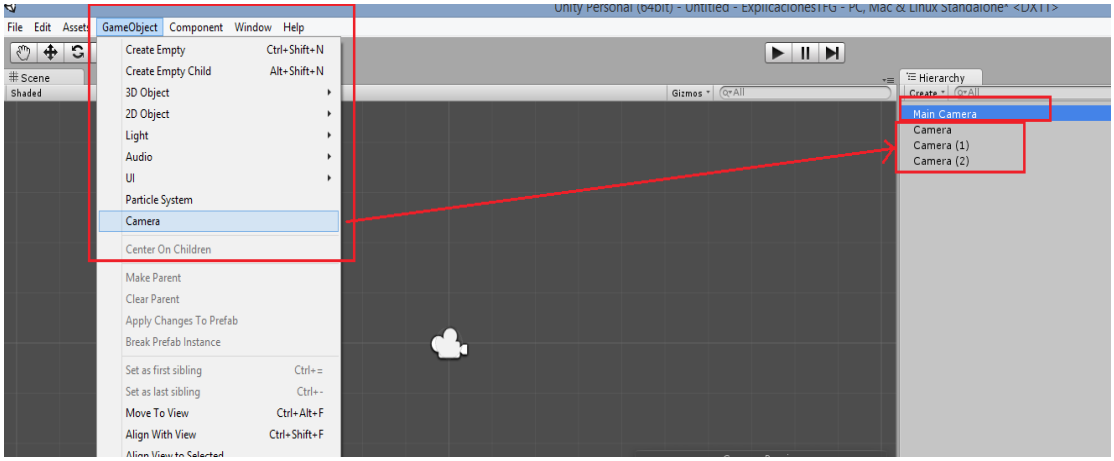


Figura 15: Creació d'un objecte càmera.

En definitiva, com qualsevol altre element, les càmeres es poden animar controlant-les mitjançant les diferents físiques del motor gràfic, modificar-les amb efectes personalitzats o combinar-les de manera que suportin multi jugador. Tot el que puguis imaginar es pot realitzar.

Configurant la càmara

Primer de tot, visualitzem en l'editor quina relació d'aspecte volem per la nostra aplicació. En aquest cas, considerem la relació 16:9, ja que és semblant a la relació d'aspecte dels dispositius mòbils, concretament, smartphones i algunes tablets del mercat actual.

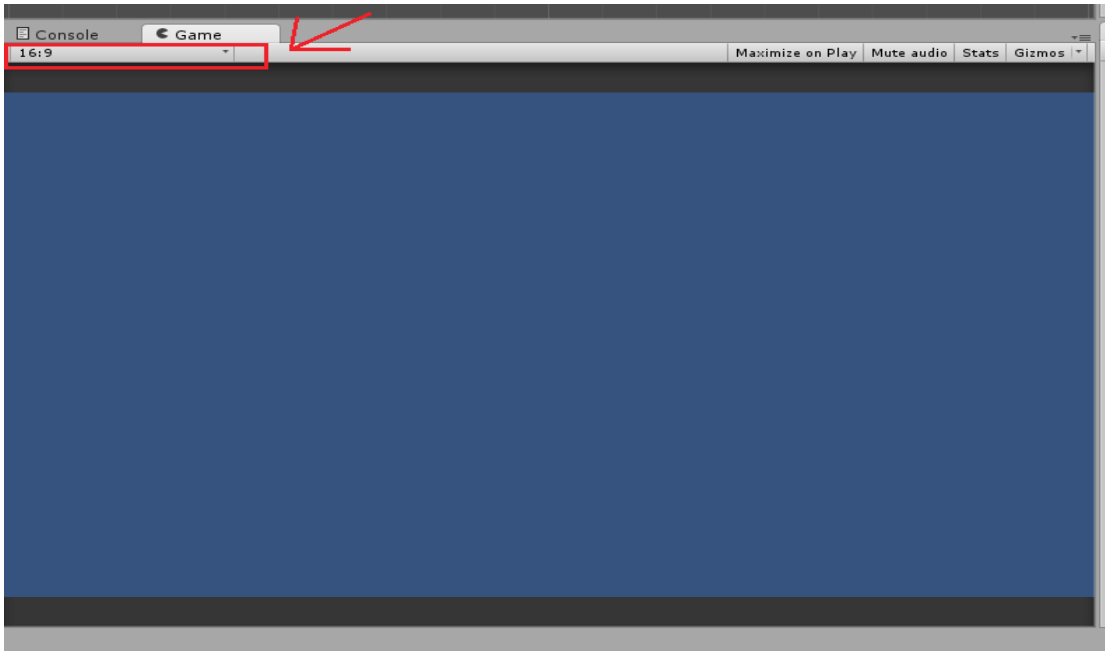


Figura 16: Assignant la relació d'aspecte a la pestanya.

Seleccionant-la en la pestanya “Game” podem visualitzar el resultat de l'aplicació, que es veurà en el dispositiu mòbil, durant el seu desenvolupament.

Finalment, només queda assignar aquesta relació a la càmera principal. Això ho farem mitjançant el desenvolupament d'un script que anomenem “AspectUtility.cs”. Per tant, primer creem la carpeta que contindrà tots els scripts de l'aplicació dins de la carpeta de recursos “Assets” i, seguidament l'script anterior dins d'aquest nou directori:

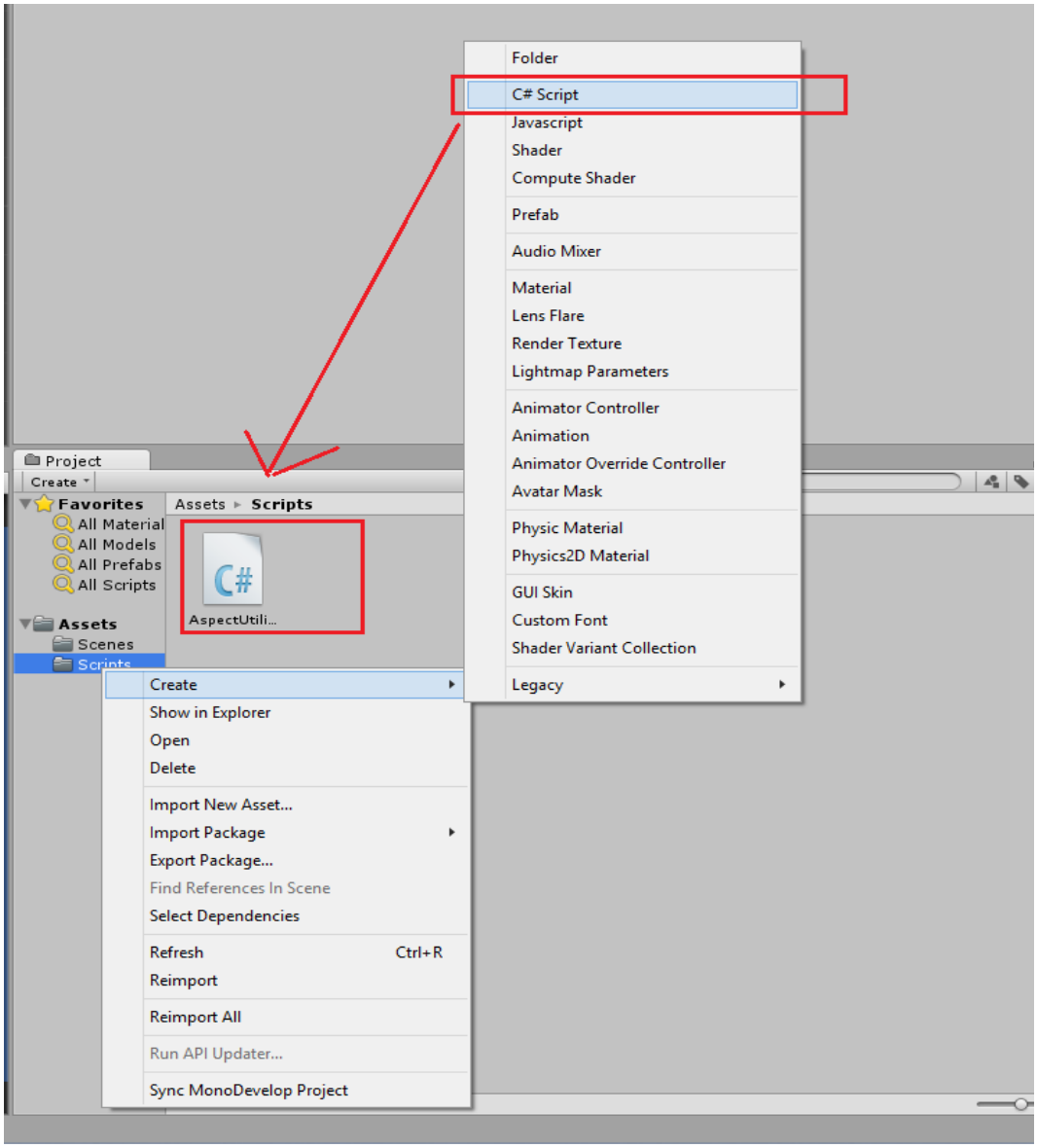


Figura 17: Creació d'un script.

Un cop desenvolupat aquest script, per assignar-lo només cal arrossegar l'arxiu fins a l'objecte corresponent. Fet això, seleccionant la càmera, apareix un nou component script a l'objecte que, en aquest cas, requereix un valor per indicar quina relació d'aspecte volem que tingui la càmera. Per calcular aquest valor cal fer la divisió 16 entre 9, degut que la relació és 16:9. Finalment indiquem el resultat en el camp corresponent:

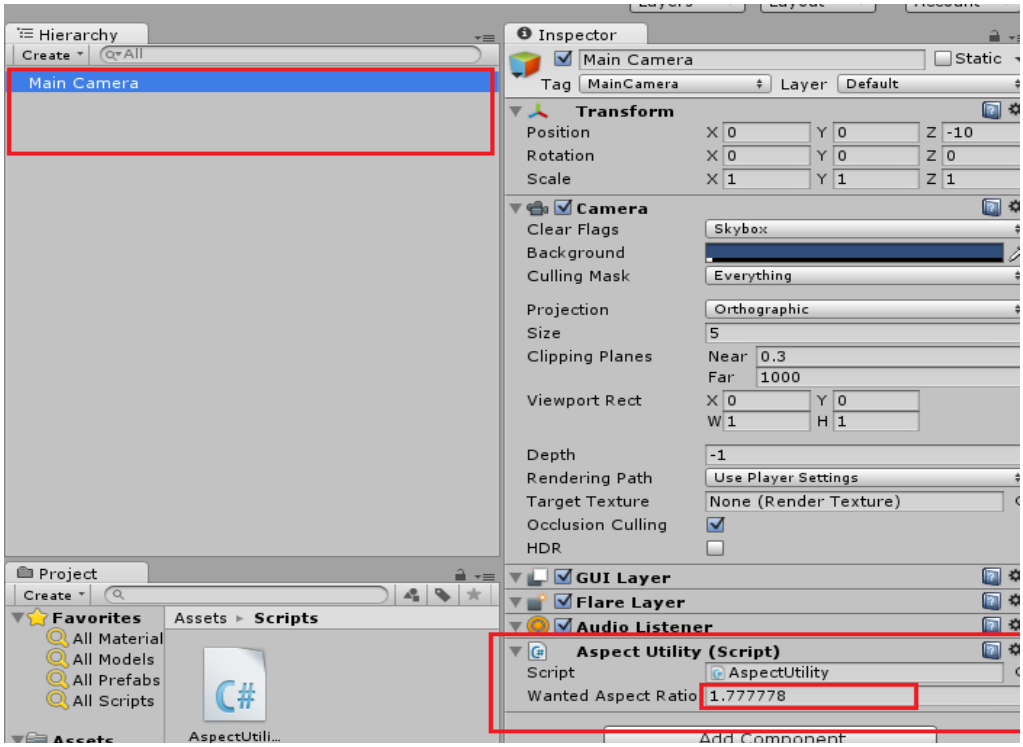


Figura 18: Assignant l'script a la càmera.

Seguint aquests passos, ja hem assignat la relació d'aspecte 16:9 a la càmera principal d'aquesta escena. Per tant, per aconseguir que tota l'aplicació mantingui aquesta relació cal assignar aquest script a totes i cada una de les càmeres.

Fent això aconseguim que el videojoc mantingui la relació d'aspecte 16:9 sense importar la resolució del dispositiu mòbil. És a dir, la imatge s'adaptarà segons la mida de la pantalla.

6.3. Definint el menú principal

Un cop configurada la primera càmera, ja podem començar a desenvolupar el menú principal.

Menú principal

El menú principal és un llistat d'opcions presentades, normalment, mitjançant text, imatges o botons en el qual l'usuari pot escollir quina tasca executar.

Iniciant la navegació des del menú principal s'ha de poder arribar a qualsevol funcionalitat que ofereixi l'aplicació. Per aquesta raó, entre altres, és una de les primeres pantalles que es mostra a l'usuari a l'hora d'executar l'aplicació.

Les opcions més usuals que podem trobar en el menú principal d'un videojoc són:

- **Jugar**: És l'opció principal d'un videojoc i permet a l'usuari iniciar o continuar la seva partida.

- **Opcions**: En aquesta opció es recull un seguit de característiques del joc els quals l'usuari pot modificar per adaptar l'experiència al seu gust com, per exemple, la dificultat, volum del so, etc.

- **Rècords:** Opció on l'usuari pot veure la seva màxima puntuació o assoliments aconseguits. També els pot comparar amb altres jugadors en línia.

- **Sortir:** És una funció la qual tanca l'aplicació completament.

Obviament, existeix moltes més opcions a oferir com xarxes socials, informació, etc. Nosaltres hem definit les següents:

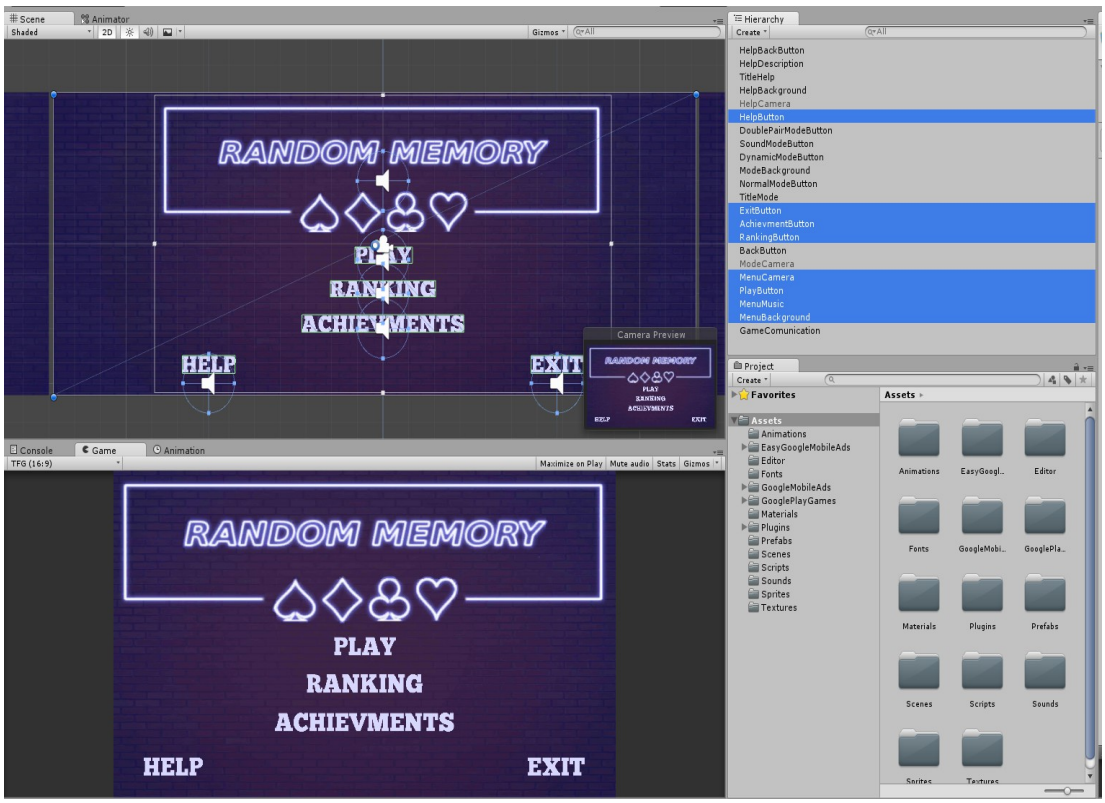


Figura 19: Components del menú principal.

On els objectes que el composen són els que estan seleccionats en l'inspector. Aquest menú està composat per:

- **Botons:** Es tracten d'objectes de text 3D animats de manera que simulin opcions que l'usuari pugui seleccionar.

- **Fons de pantalla:** Es tracta d'un objecte de tipus Quad que conté el material amb la imatge que es mostra en la captura.

- **Música:** Es tracta d'un objecte buit que conté el component que reprodueix l'arxiu de música assignat.

6.3.1. Botons

Els botons d'una aplicació ens permeten navegar per diferents menús. Per tant, es tracta d'un dels elements bàsics en qualsevol video joc.

Primerament, per definir un botó creem un objecte de text en 3D i el posicionem i indiquem el tamany assignant els valors al component *Transform*. Per exemple, el botó jugar:

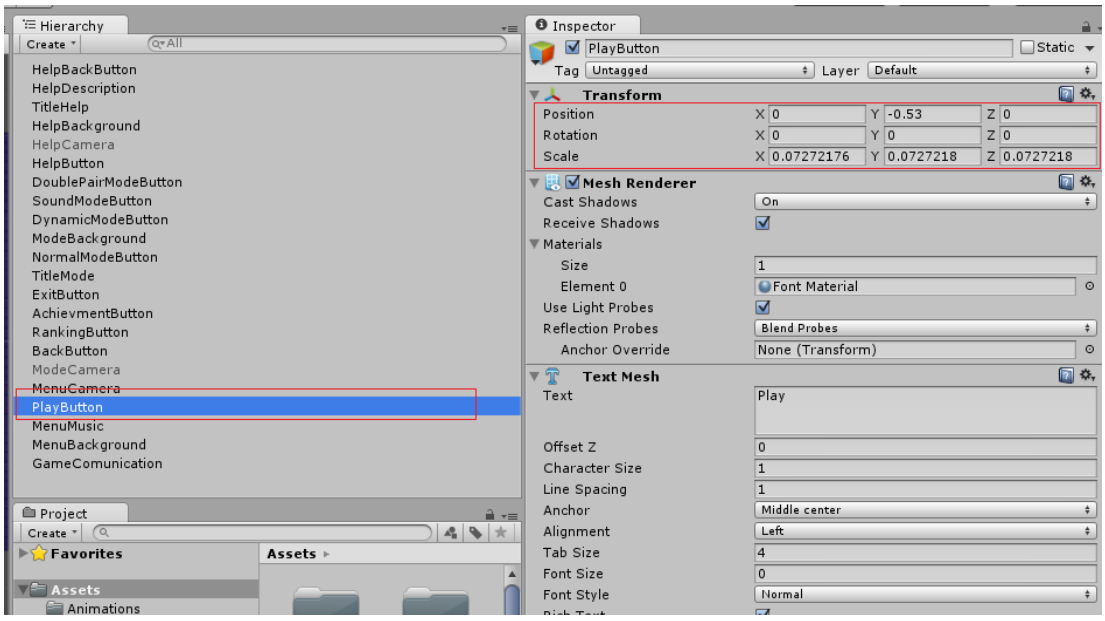


Figura 20: Posició i tamany del botó jugar.

Seguidament, podem assignar una font al component *TextMesh* de l'objecte per aplicar un estil visual al text. En aquest cas assignem:

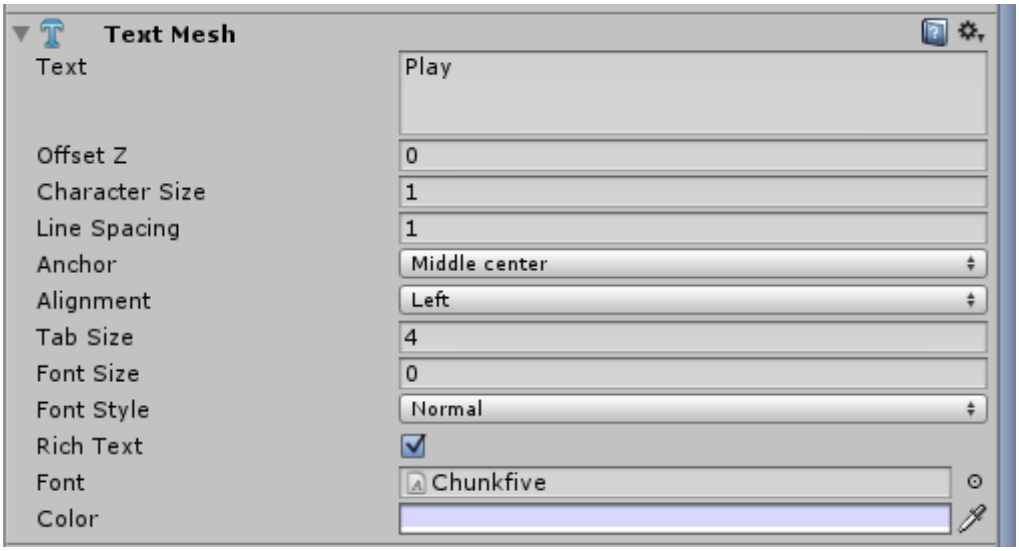


Figura 21: Component *TextMesh* del botó jugar.

En aquest component definim, entre altres:

- **Text:** Aquí definim la cadena de text que volem que es mostri per pantalla.
- ***Anchor*:** És el punt en el que el sistema es basarà a l'hora de posicionar l'element a la pantalla.
- ***Alignment*:** És la direcció on s'afegirà el nou text, en cas de modificar la cadena de text.
- **Font:** És l'arxiu font que assignem al text, en aquest cas *Chunkfive*.

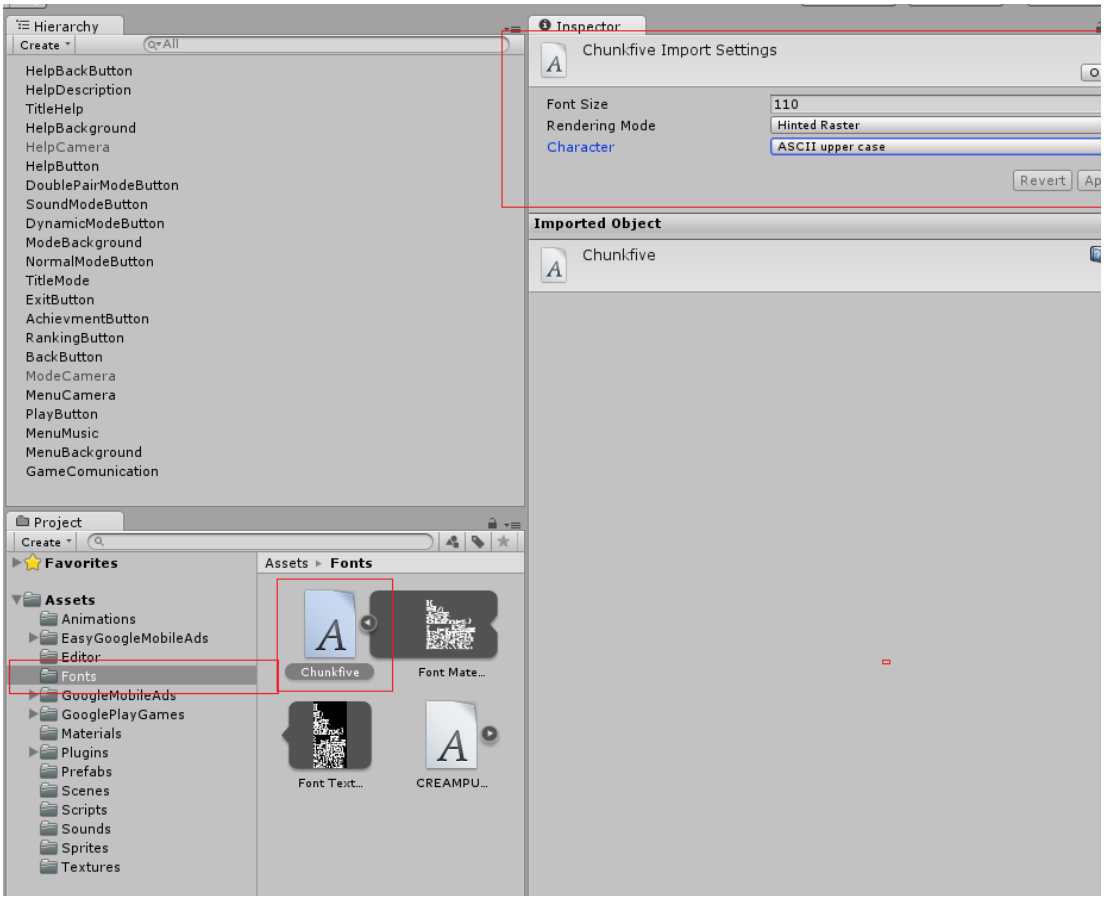


Figura 22: Configuració de la font.

Amb aquestes modificacions el botó es veu com a les imatges anteriors. Ara cal definir la comunicació entre l'usuari i l'aplicació. Per a que el text detecti que l'usuari ha tocat l'opció, cal afegir un *Collider* a l'element, en aquest cas un *Box Collider*.

Collider

Un *collider* no és més que un component que detecta les col·lisions entre elements o entre usuari i element. Per exemple, si tenim un personatge que salta i cau a terra, els dos elements han de tenir assignats *colliders* per detectar la col·lisió i no transpassar l'element, mitjançant configuracions i scripts.

En el nostre cas, quan l'usuari toca la pantalla en el punt (x,y), si aquest punt es troba dins del detector de col·lisions, aquest el detectarà i, mitjançant scripts realitzarà la funció per canviar al menú de selecció de modes.

Per tant, per assignar el detector de col·lisions primer afegim el component a l'objecte i seguidament assignem el tamany i posició corresponents al text:

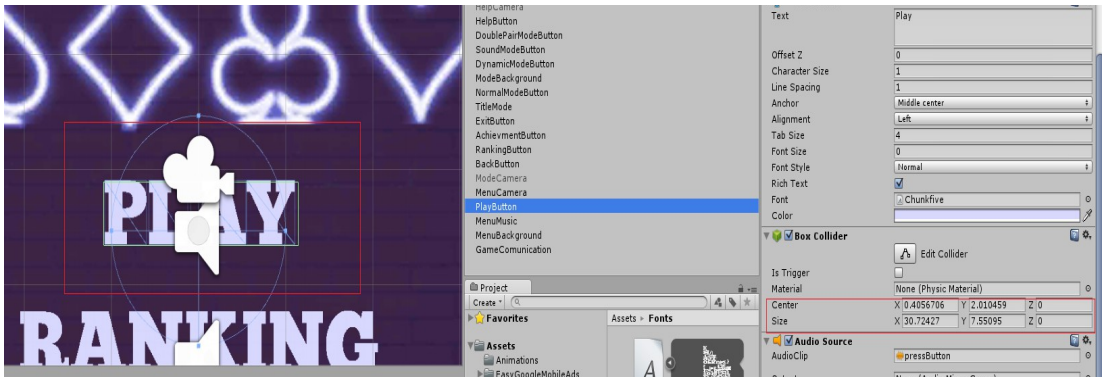


Figura 23: Assignació del *Box Collider*.

Un cop tenim el *collider*, el sistema ja pot detectar quan l'usuari selecciona el botó. Ara, per notificar a l'usuari que el sistema ha detectat el seu dit sobre el botó, cal afegir un component per reproduïr l'arxiu de so que assignem.

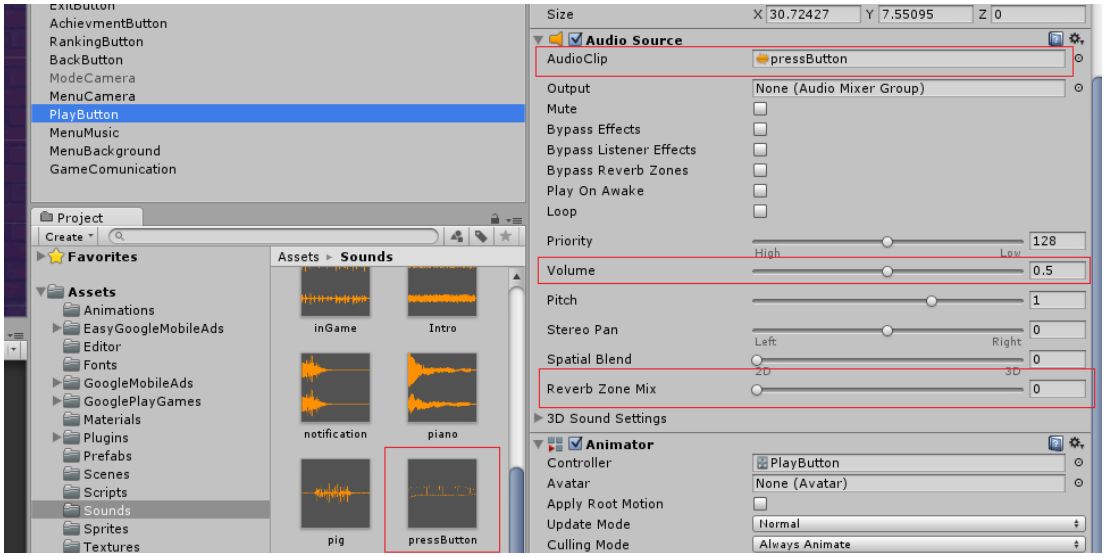


Figura 24: Configuració del so del botó.

L'arxiu d'audio que assignem es *pressButton.wav* amb el volum definit a la meitat i en 2D. La resta de variables es deixen per defecte. A l'apartat Música veurem més característiques.

Seguidament, per a que l'usuari es doni compte que el text que visualitza es seleccionable, afegim una animació a l'element.

Animacions

Una animació no és més que aplicar un moviment o transformació a un objecte per un determinat esdeveniment. En el cas dels botons, per crear el clip que contindrà l'animació:

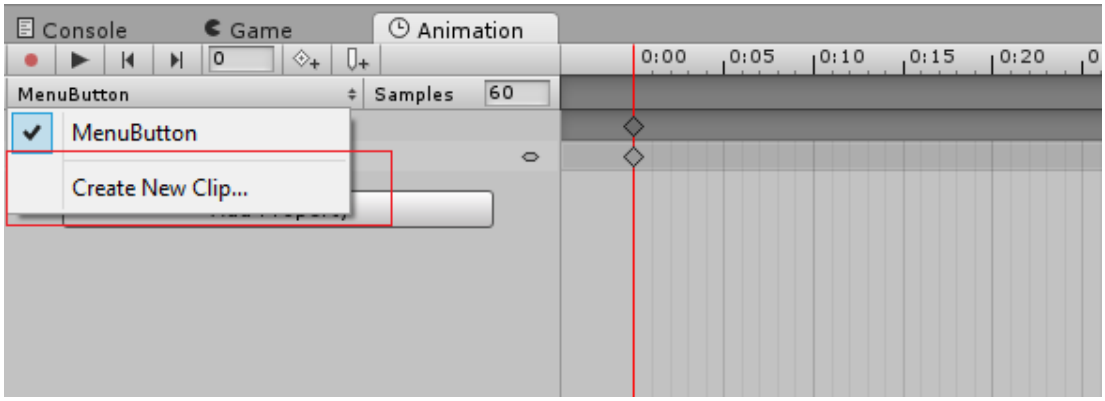


Figura 25: Creació del clip d'animació.

I el guardem a la carpeta *Animations* amb el nom de *MenuButton*. Seguidament, assignem la variable *scale* del component *Transform*, que serà la que es modificarà durant l'animació.

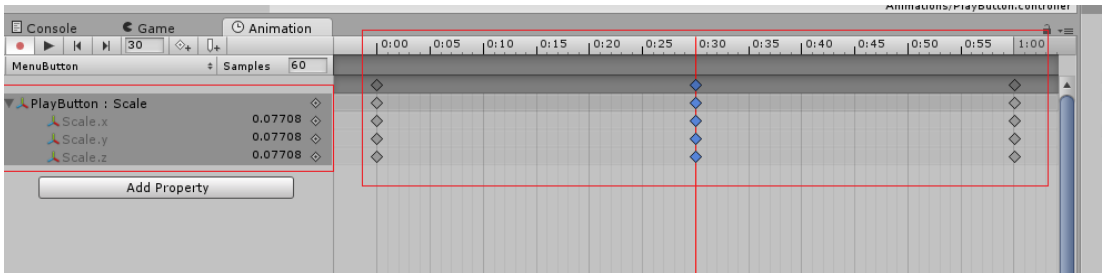


Figura 26: Creació de l'animació.

Es tracta d'una animació d'un segon de duració on al mig de l'animació fem el text més gran incrementant els valors de *scale*, de manera que a l'instant 0 es veu el text normal, incrementa el seu tamany i torna al seu estat normal. Només queda assignar l'animació a l'element, de manera que:

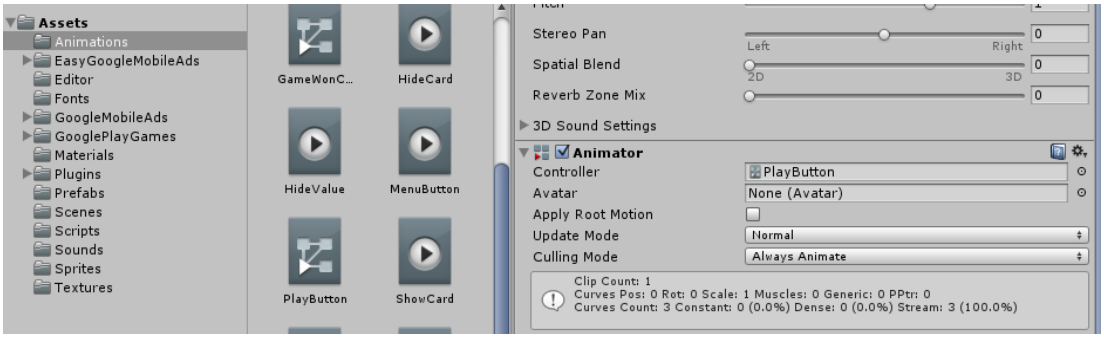


Figura 27: Assignació de l'animació al component *Animator*.

Fins aquí, tenim el botó creat però sense cap tipus de funcionalitat assignada. Per tant, cal afegir l'script corresponent a l'objecte, que s'encarregarà, en aquest cas, d'activar i desactivar les càmeres corresponents per passar del menú principal al menú de selecció de modes. Per altra banda, aquest botó és l'únic de l'aplicació que té com a valor cert la variable *Show Intersticial* de l'script, degut que és aquí on volem que es mostri publicitat quan l'usuari toca aquesta opció. Es veurà en més detall més endavant.

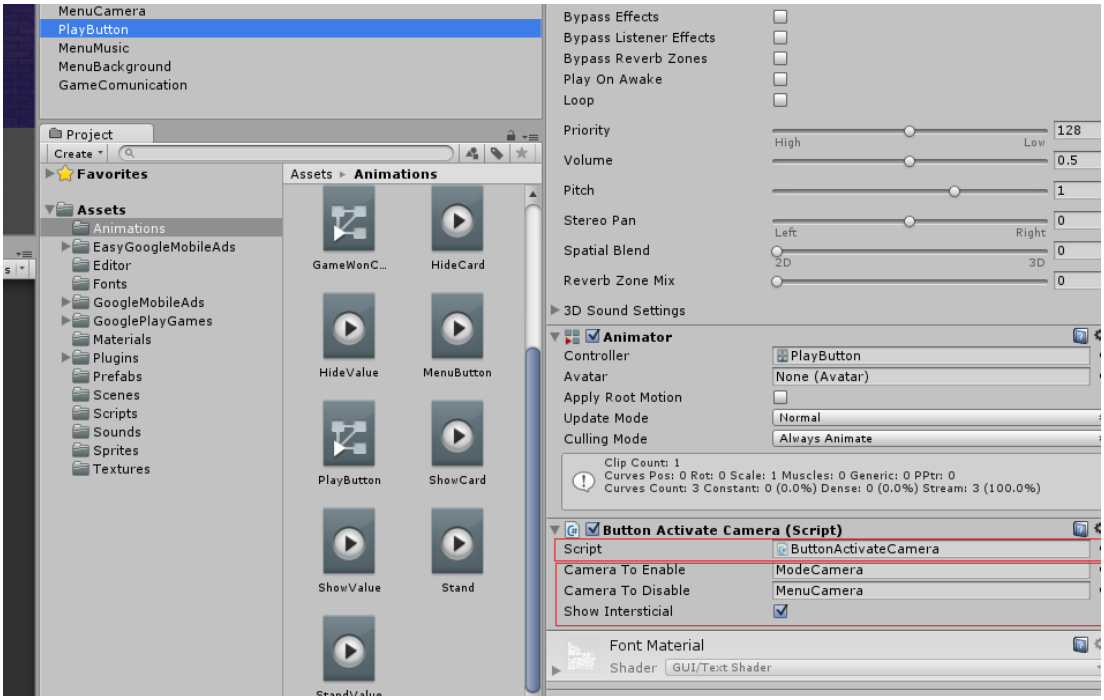


Figura 28: Assignació de l'script.

Completant els passos anteriors, obtenim el botó tal i com es presenta en l'aplicació, amb l'estil visual i funcionalitat corresponent. Per tant, aplicarem la mateixa metodologia a qualsevol dels botons de l'aplicació però amb funcionalitats diferents assignant diferents scripts.

6.3.2. Fons de pantalla

Per qualsevol fons de pantalla, sigui del menú o sigui de l'escenari cal crear un objecte *Quad*, on en el seu component *Mesh Renderer* cal assignar el material que forma el la imatge del fons.

Materials

Es tracten de definicions sobre una superfície. Conté diferents característiques configurables segons el *shader* utilitzat.

Shaders

Són scripts amb càlculs i algoritmes que calculen el color de cada pixel segons la configuració del material.

Textures

Són les imatges de mapa de bits.

Un cop definits breument aquests tres conceptes, primer incorporem la imatge que volem de fons de pantalla del menú a la carpeta *Textures* i la configurem de la manera següent:

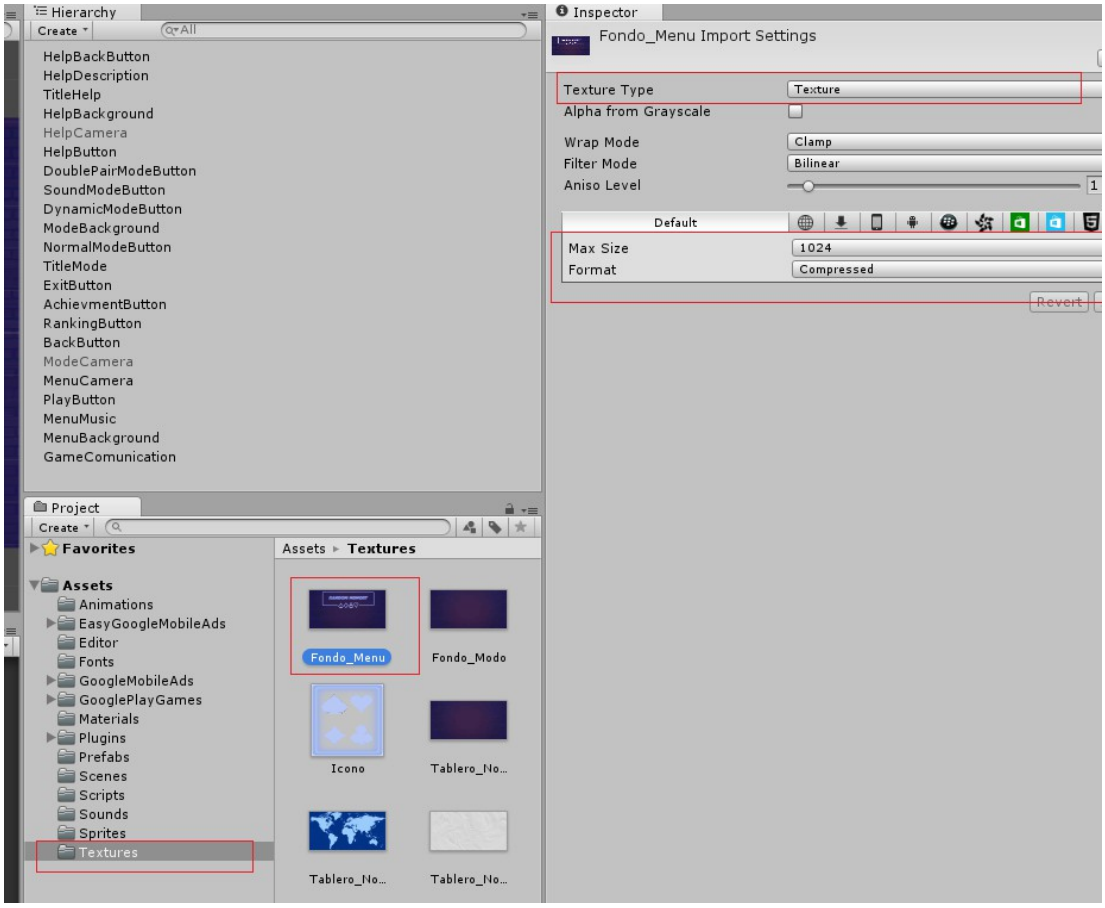


Figura 29: Creació de textures.

Indiquem que es tracta d'una textura, i volem que el màxim tamany sigui 1024 i amb format comprimit, degut que no volem que ocupi gaire espai en memòria. Aquesta configuració s'aplicarà a totes les textures del joc (fons de menú, fons d'escenaris, icones, etc).

Seguidament, cal crear el material que posteriorment assignarem al component de l'objecte. Per tant, a la carpeta Materials creem un nou material, indiquem que el *Shader* és de tipus *unlit/texture* i seleccionem la textura anterior:

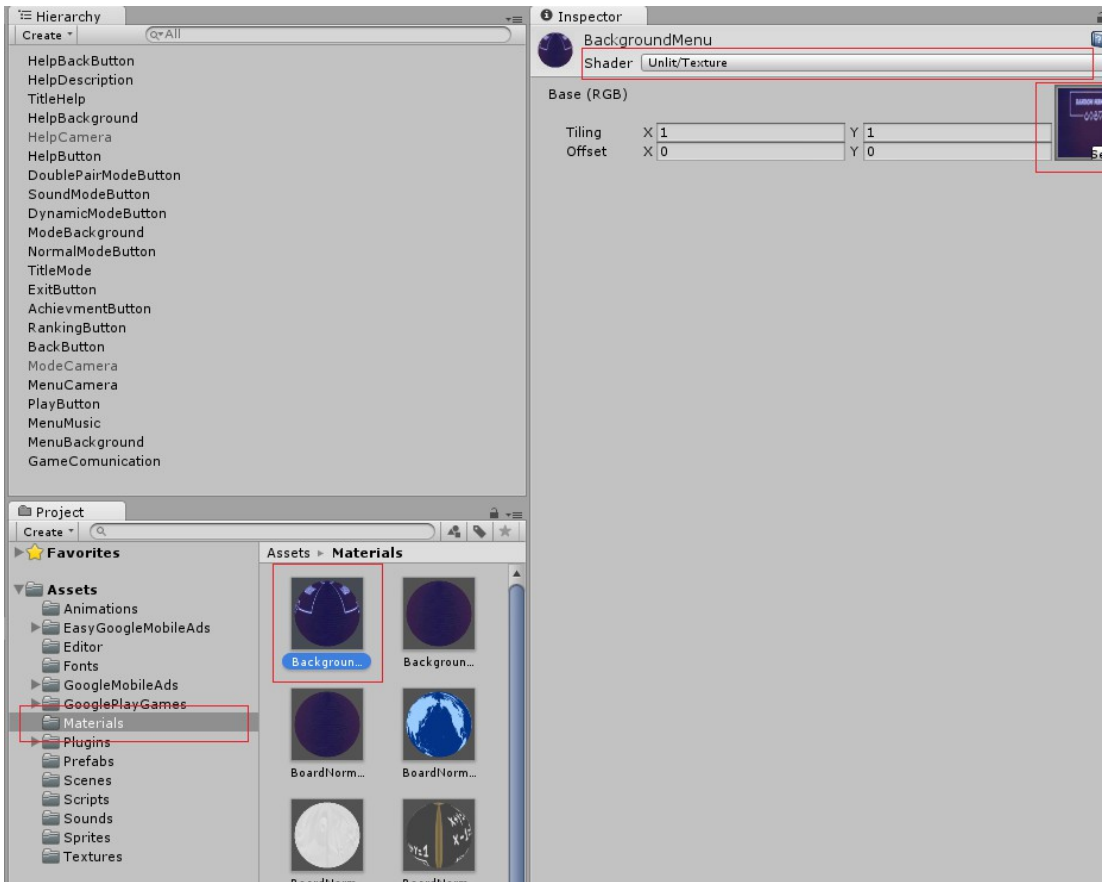


Figura 30: Creació de materials.

Un cop creat el material, ja es pot afegir al component de l'objecte que conté el fons de pantalla. Posteriorment cal ajustar el tamany i posició amb el component *Transform*, ja vist anteriorment. Per tant:

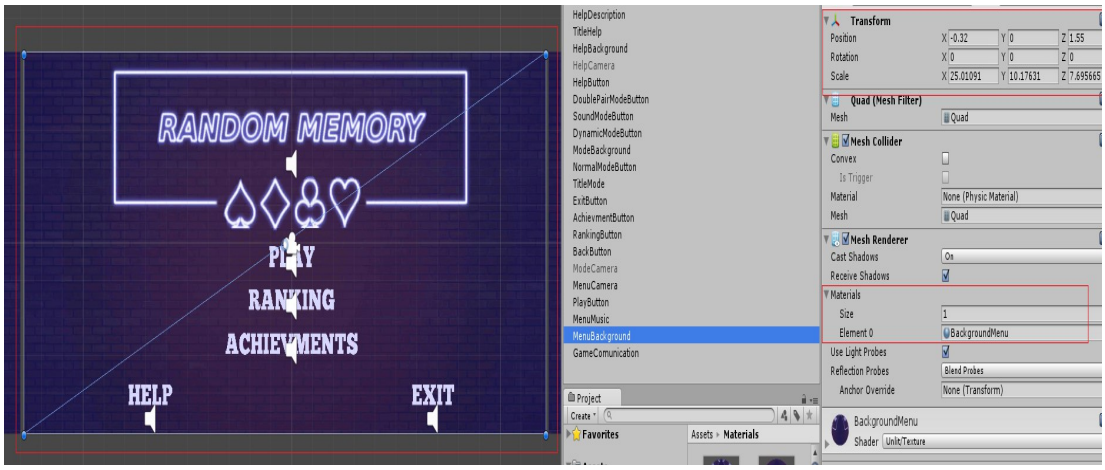


Figura 31: Assignació del material.

Amb aquests passos, podem tenir qualsevol fons de pantalla de totes les escenes, tal i com es presenten en l'aplicació.

6.3.3. Música

En tot videojoc ha d'haver música de fons, ja sigui música o sons ambientals segons el tipus de joc. En aquest projecte, per aplicar música de fons als menús creem un objecte buit i el posicionem en qualsevol lloc de l'entorn, degut que aquest no es visualitzarà per pantalla.

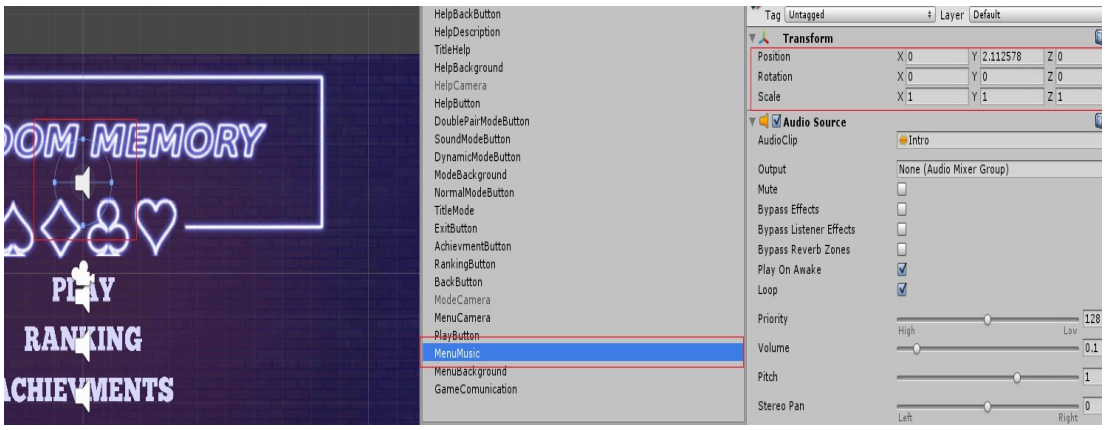


Figura 32: Creació de la música del menú.

Seguidament, cal afegir l'arxiu de so que es reproduirà durant la navegació pels menús, com hem vist anteriorment. En aquest cas, ens interessa que la música comenci només al carregar el menú principal i que es vagi repetint infinitament fins començar la partida. Això s'indica a les característiques del component, de manera que:

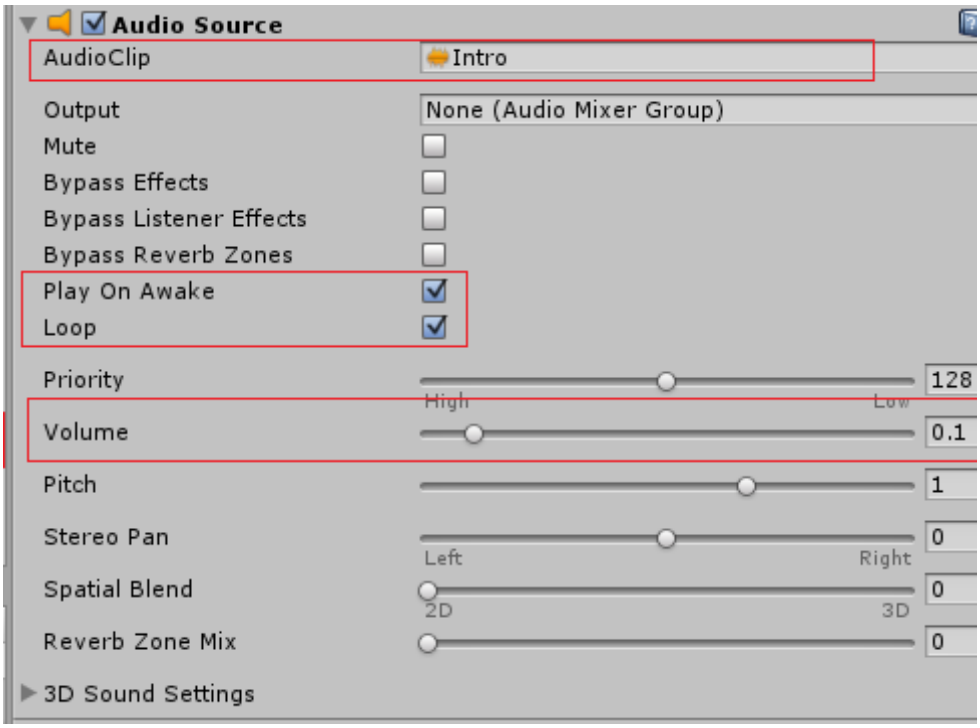


Figura 33: Configuració del so.

- ***Play on Awake:*** Indica si inicia la reproducció al construir l'element en pantalla.

- ***Loop:*** Indica si volem que es repeteixi infinitament la música fins la destrucció de l'objecte que la conté.

Amb aquests senzills passos, aconseguim tenir música al menú principal, tal i com es presenta a l'aplicació.

6.4. Pantalla d'ajuda

Aquesta pantalla resulta ser molt simple en comparació al menú principal. Està formada per:

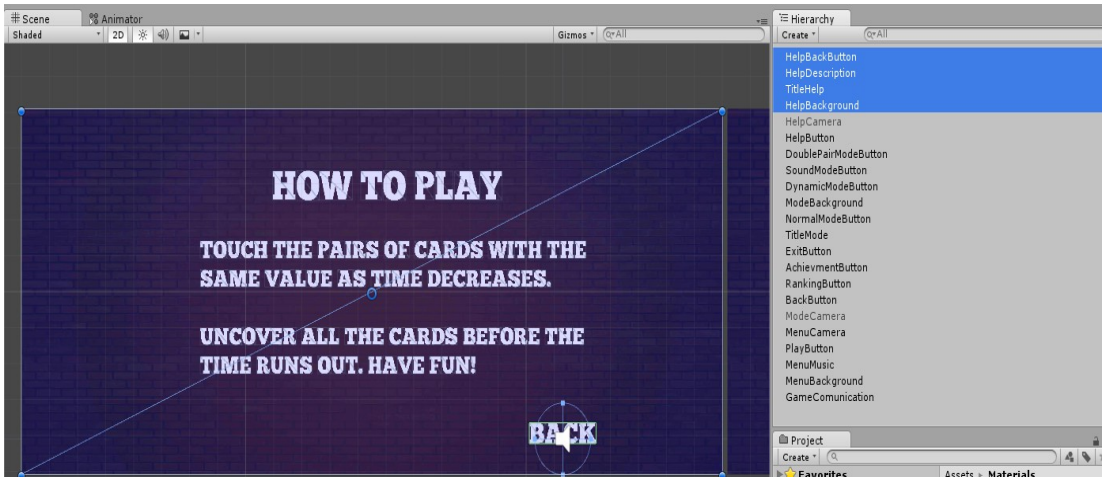


Figura 34: Components de la pantalla d'ajuda.

- **Botó tornar:** Definit amb els mateixos components que els anteriors amb la diferència que les càmeres que s'activen i desactiven canvien a la inversa.
- **Descripció:** Text 3D amb la mateixa font que els botons.
- **Títol:** Igual que l'anterior, però amb diferent tamany.
- **Fons de pantalla:** Objecte *Quad* idèntic al fons de pantalla del menú principal.

L'objectiu principal d'aquesta pantalla és mostrar una idea mínima de com s'ha de jugar la partida al joc amb una descripció breu i resumida.

6.5. Selecció de mode de joc

Aquesta pantalla és un menú secundari format per cinc botons, un títol i un fons de pantalla on l'usuari podrà seleccionar el mode de joc en el que vol iniciar la partida.

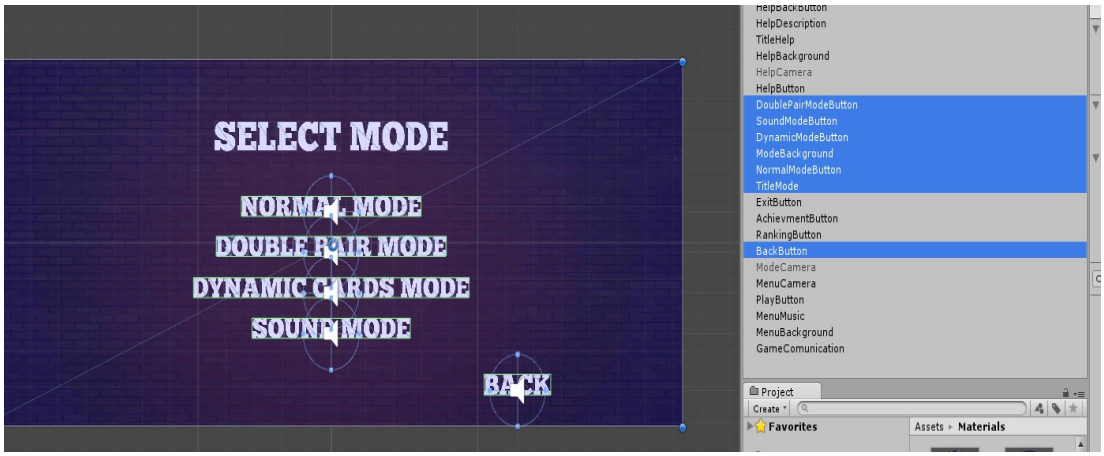


Figura 35: Components de la pantalla de selecció de mode.

La diferència principal amb els botons anteriors, és que aquests contenen un altre script. En aquest script, afegim el nom de l'escena que volem iniciar. Per exemple, el botó del mode normal:

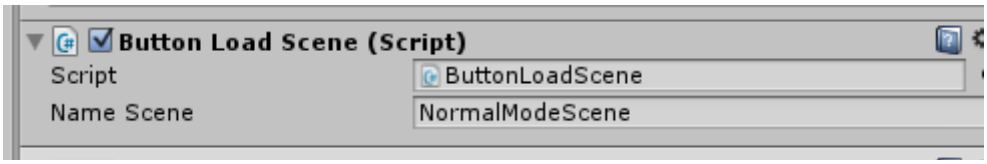


Figura 36: Script per la selecció de modes.

Scenes

Una escena és on es guarden tots els elements de les parts de l'aplicació. Per exemple, en una película existeix més d'una escena que conté personatges, converses, música, llocs, etc.

Una aplicació ha de tenir, almenys, una escena però pot tenir les que facin falta. Nosaltres definim cinc escenes en aquesta aplicació: una per els menús, que conté el menú principal, menú de selecció de modes i la pantalla d'ajuda, i els altres quatres corresponen als quatre modes de joc que hi ha. Per guardar una escena només cal anar a File -> Save Scene as -> Scenes -> Nom de l'escena.

Per tant, un cop l'usuari prèmi una de les quatre opcions, l'escena corresponent iniciarà i així començarà la partida corresponent.

6.6. Implementant el mode normal

Aquesta és l'escena que reproduïrà la partida en el mode de joc normal. Per tant, conté els elements principals de l'aplicació que anirem explicant en els següents sub apartats.

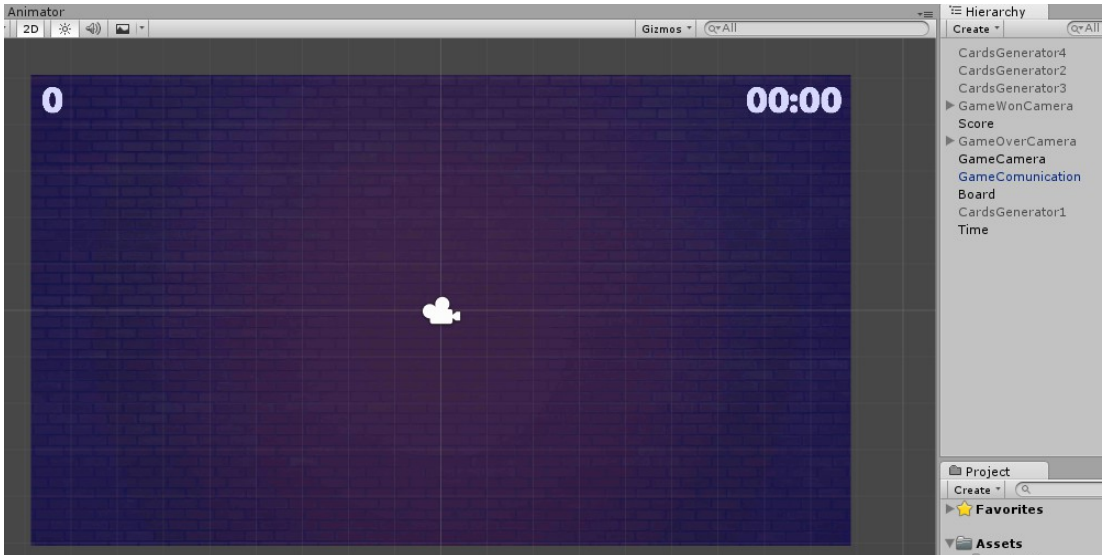


Figura 37: Components del mode normal.

6.6.1. Tauler aleatori

El que volem aconseguir és que cada cop que l'usuari inicií una partida en qualsevol dels tres primers modes, aquest mostri un escenari aleatori.

Per tant, creem un fons de pantalla com hem vist anteriorment, però amb la diferència que assignem l'script encarregat de gestionar l'escenari que es mostra, juntament amb el color del marcador, de la puntuació i el tipus de cartes que es mostren. Per aquest motiu, asignem els objectes corresponents:

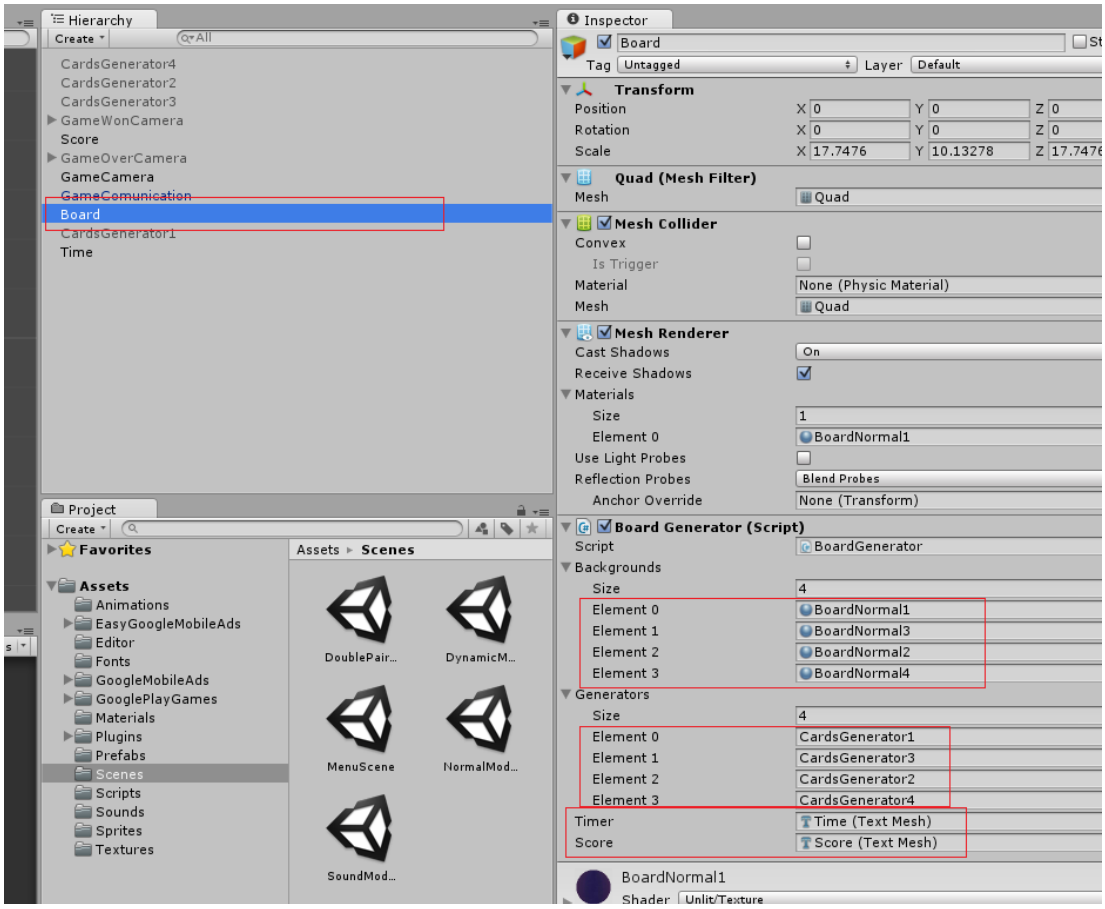


Figura 38: Components del mode normal.

D'aquesta manera, aconseguim que segons l'escenari triat aleatoriament, mostri la resta de components corresponents.

6.6.2. Temporitzador

El temporitzador limita a un temps determinat la duració de la partida, en quant el temps finalitzi, el jugador perd la partida.

L'element que representarà el temps per pantalla és un altre text en 3D, però assignant l'script responsable de gestionar el temps de la partida.

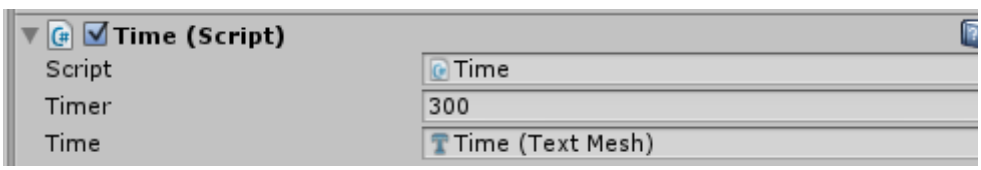


Figura 39: Script que gestiona el temporitzador.

En aquest cas, cal assignar la duració del temps en segons juntament amb el component *Text Mesh* del propi objecte, ja que s'anirà actualitzant la cadena de text des de l'script.

6.6.3. Puntuació

El marcador de puntuació indica a l'usuari quina és la puntuació que va aconseguint durant la partida. Per tant, com el cas anterior, es tracta d'un altre text en 3D però assignant l'script encarregat de gestionar el sistema de puntuació del joc.

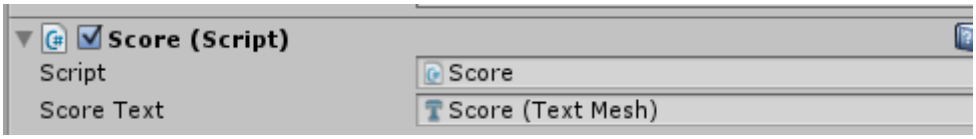


Figura 40: Script que gestiona el sistema de puntuació.

En aquest cas, cal assignar el component *Text Mesh* del propi objecte, ja que s'anirà actualitzant la cadena de text des de l'script.

6.6.4. Generador de cartes

El generador de cartes és el responsable de crear i mostrar les cartes en la pantalla. Es tracta d'un objecte buit amb l'script assignat per tal de poder realitzar la seva funcionalitat principal.

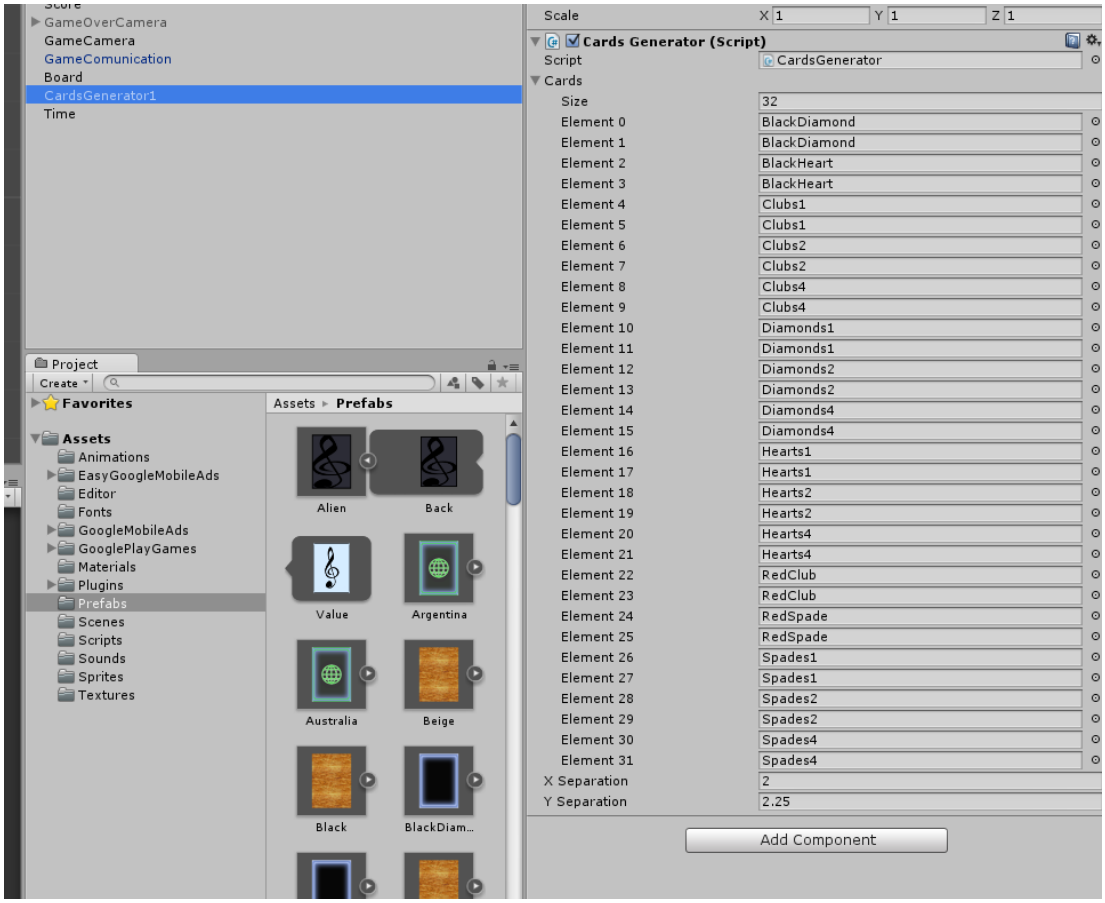


Figura 41: Cartes assignades a un generador de cartes.

En aquest cas, cal afegir les cartes prefabricades que volem que es mostrin en la partida per duplicat, ja que aquest mode tracta d'endevinar parelles de cartes, juntament amb la separació que hi ha d'haver entre elles. Cada generador correspon a un tipus d'escenari. Per tant, cal crear nous generadors per nous escenaris assignant les cartes corresponents.

Prefabs

Un objecte prefabricat no és més que un conjunt d'elements configurats de manera que es puguin crear les còpies necessàries de l'objecte exactament amb les mateixes característiques.

En la nostra aplicació, tenim com a objectes prefabricats totes les cartes del joc, ja que es crearan durant la partida diverses còpies d'aquestes. Per crear una carta tenim, per exemple:

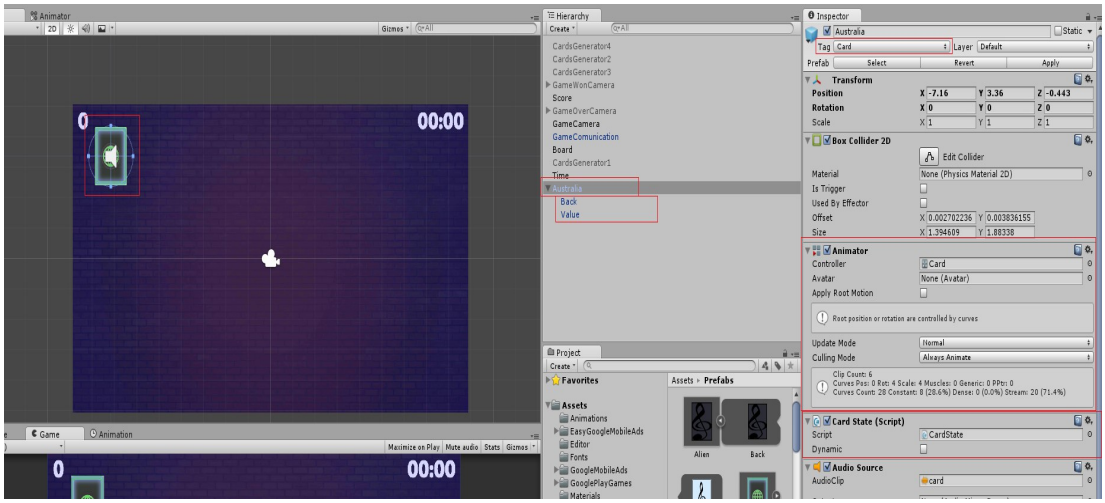


Figura 42: Creació d'una carta

Aquesta carta Austràlia és un objecte buit compost per dos objectes de tipus *Sprite*, un conté la imatge de la tapa de la carta i l'altre el valor de la carta. Per tant, aquests dos han de tenir exactament el mateix tamany i posició, però amb la diferència que el valor de la carta ha d'estar girada 180 graus, degut que la carta quan mostri el valor ha de donar la volta.

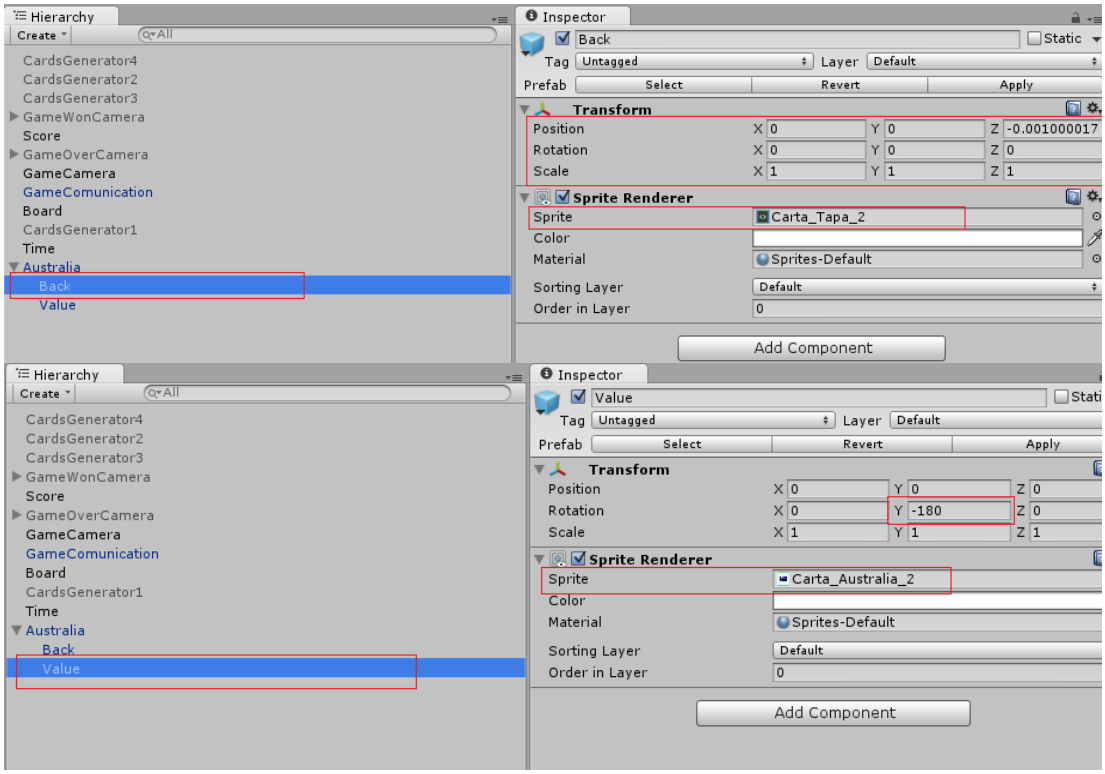


Figura 43: Posicions dels *sprites* en la carta.

Sprites

Un sprite és un tipus de recurs visual que tindrà diferents accions en la partida. En el cas de les cartes, poden ser seleccionades i tenen diferents animacions. Com en el cas de les textures, tenen una configuració similar.

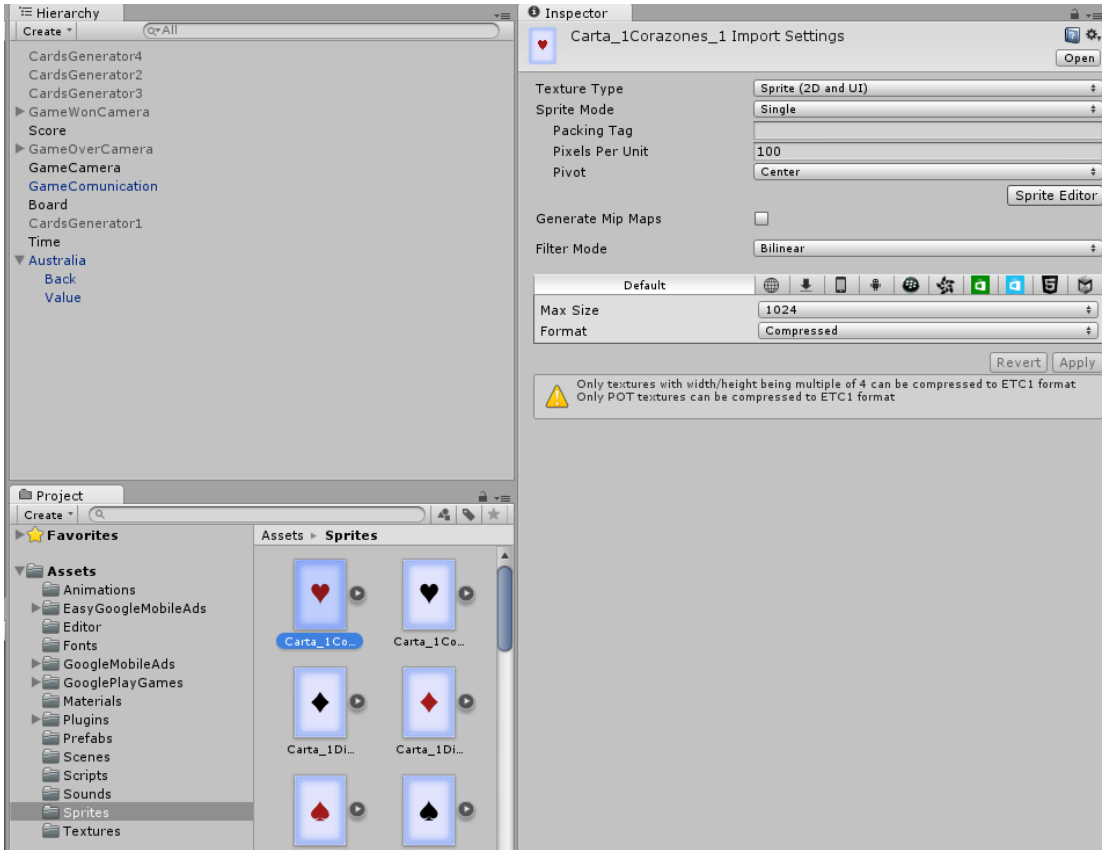


Figura 44: Creació d'un *sprite*.

Com es pot veure en la figura 42, també afegim l'etiqueta *Card* a l'objecte, degut que mitjatçant scripts d'altres elements podrem detectar quines cartes ens interessen segons l'operació a realitzar. També afegim l'script encarregat de gestionar l'estat i les animacions de la carta durant la partida.

Pel que fa l'animació de la carta, hem d'assignar una sèrie de transaccions per gestionar l'estat de l'animació:

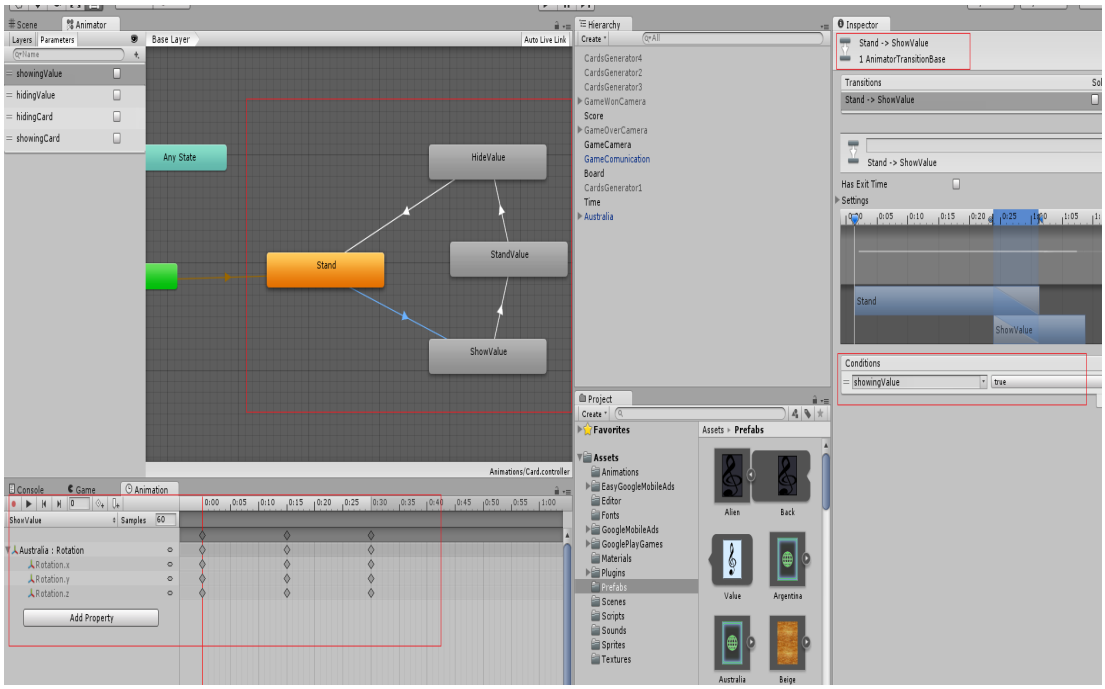


Figura 45: Animació d'una carta.

Cada una de les fletxes indiquen una transacció entre dos estats de l'animació, en el cas de la figura anterior, tenim que quan el valor de *showingValue* sigui cert l'estat de l'animació canviarà de *Stand* a *ShowValue,* que farà que la carta tingui una rotació de 180 graus per mostrar el valor de la carta per pantalla. La resta completen l'animació que realitzarà qualsevol carta durant la partida.

Amb aquest passos resumits, podem tenir qualsevol generador de cartes de qualsevol tipus d'escenari i de mode de joc.

6.6.5. Càmeres de la partida

Com hem vist anteriorment, qualsevol de les càmeres del joc tenen una configuració bàsica per reproduir el contingut del joc per pantalla. No obstant, les pròximes càmeres presenten petites diferències a les anteriors les quals comentarem en els següents sub apartats.

6.6.5.1. Càmera principal

Aquesta serà la càmera encarregada de reproduir el contingut de la partida. Les diferències principals amb la resta de càmeres són:

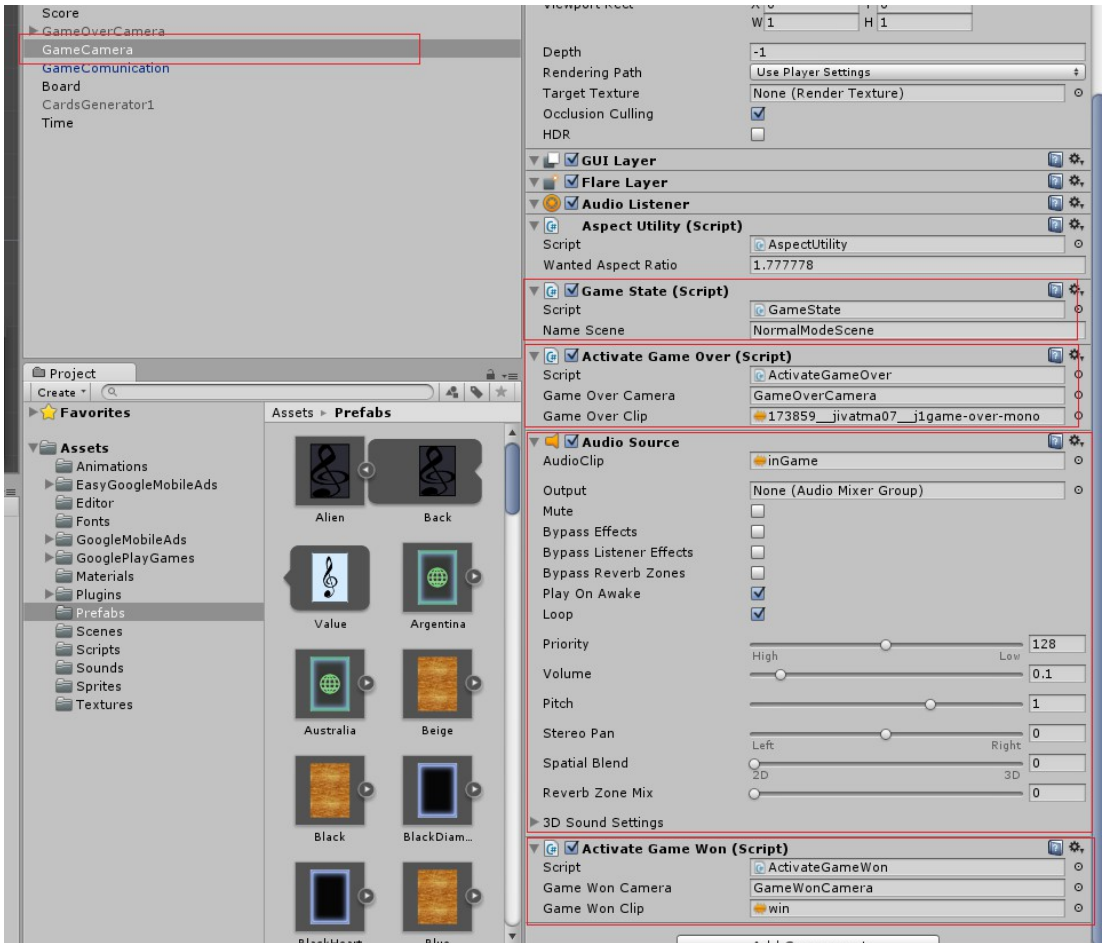


Figura 46: Càmera principal de la partida.

- Un script que gestionarà l'estat de la partida, segons les accions de l'usuari. Ha d'incloure el nom de l'escena carregada, ja que segons el mode seleccionat cal aplicar unes normes o unes altres.

- Un script que activarà la càmera de derrota. Cal especificar la càmera objectiu per ser activada juntament amb l'arxiu de so a reproduïr.

- Un arxiu de so que es reproduirà durant la partida com música de fons.

- Un últim script que activarà la càmera de victòria. Cal especificar la càmera objectiu per ser activada juntament amb l'arxiu de so a reproduïr.

Amb aquestes característiques tenim qualsevol càmera principal de les partides dels modes de joc.

6.6.5.2. Càmera de derrota

Aquesta serà la càmera que s'activarà si el temps de la partida finalitza abans que l'usuari endevini totes les cartes. Aquesta es diferència amb les anteriors degut que conté una animació que es reproduirà un cop activat l'objecte.

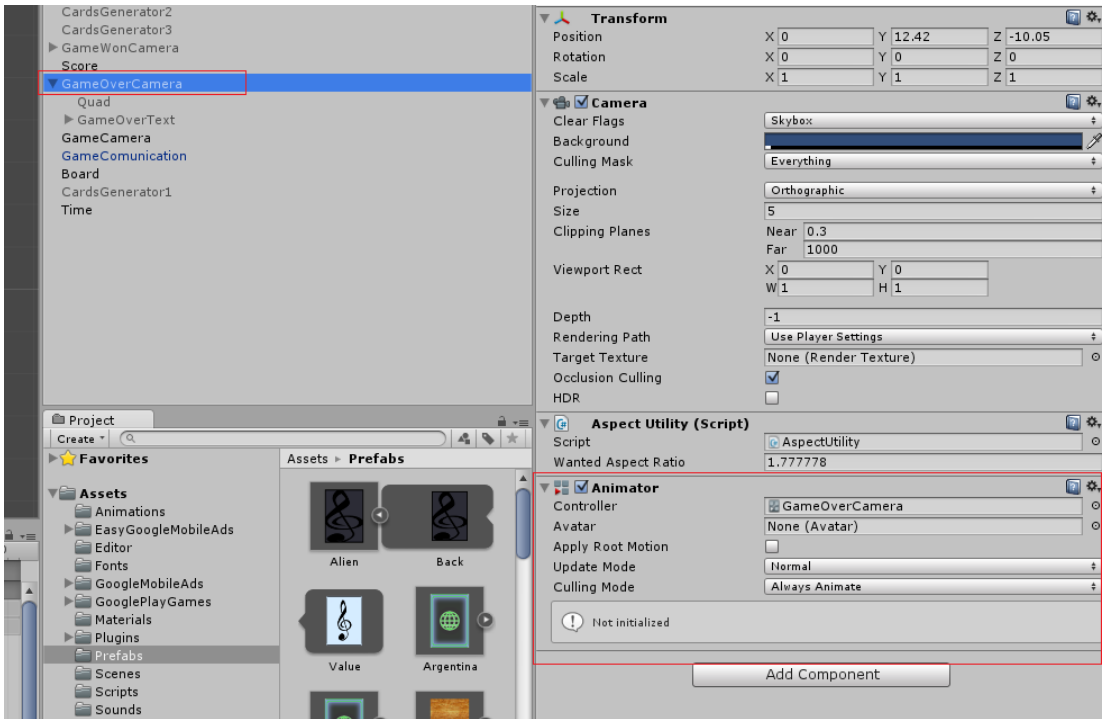


Figura 47: Càmera de derrota amb animació.

També està composta per un objecte *Quad* amb fons negre, diversos texts que contenen els missatges que es mostraran per pantalla i els botons corresponents.

6.6.5.3. Càmera de victòria

Aquesta serà la càmera que s'activarà si l'usuari completa la partida abans que el temps finalitzi. Aquesta es diferència amb l'anterior degut que conté un script que mostrarà els resultats aconseguits per l'usuari, juntament amb el seu rècord personal.

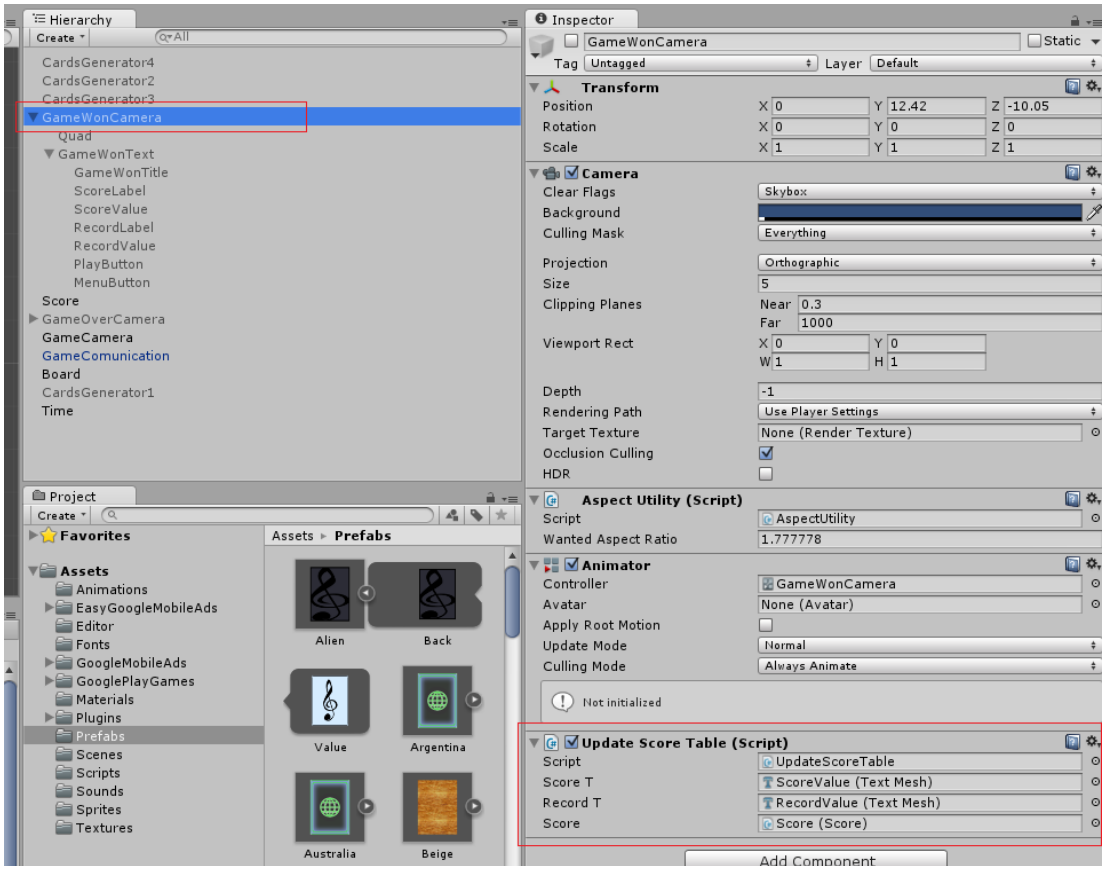


Figura 48: Càmera de victòria amb script.

També conté els mateixos components que la càmera anterior, amb petites diferències de text.

6.6.6. Comunicació entre escenes

En qualsevol aplicació és necessari comunicar certa informació entre diferents escenes. En aquest cas, cal comunicar la puntuació aconseguida per l'usuari, seguidament dels assoliments i classificacions aconseguits.

Per tant, cal crear un objecte buit amb l'script encarregat de gestionar la comunicació tant interna com externa. Aquest objecte, mitjançant la seva funcionalitat serà indestructible i estàtic, de manera que no es puguin modificar els valors assignats.

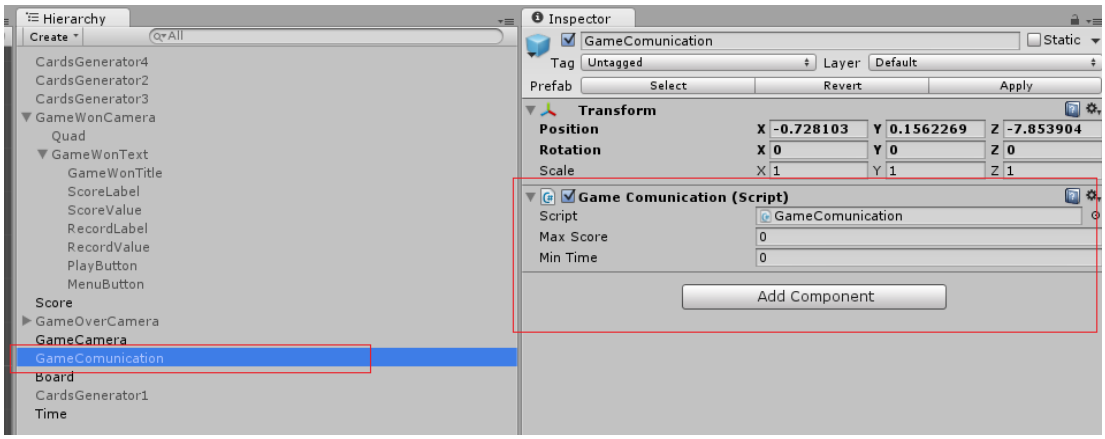


Figura 49: Objecte responsable de la comunicació de l'aplicació.

Amb tots els components definits en aquest i apartats anteriors, tenim el mode normal de joc complet.

6.7. Implementant la resta de modes

Com que els altres tres modes tenen característiques molt similars a l'anterior, les diferències a explicar seran mínimes, degut que la normativa de joc no canvia gaire.

6.7.1. Mode de joc de doble parella

La diferència principal entre aquest i la resta de modes es troba únicament en els generadors de cartes. Com que en aquest mode l'usuari ha de seleccionar quatre cartes de mateix valor, cal assignar les cartes per quadruplicat, de manera que existiran grups de quatre cartes amb valors idèntics.

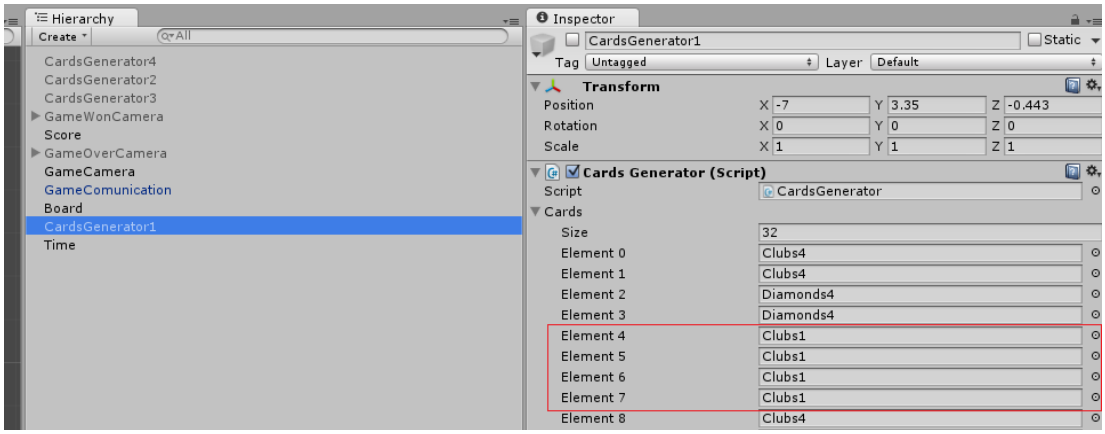


Figura 50: Cartes assignades al generador del mode de doble parella.

També cal indicar que s'ha d'actualitzar el nom de l'escena a l'script corresponent assignat a la càmera principal per aplicar la normativa del mode de joc corresponent. La resta d'elements es mantenen igual que en el mode anterior.

6.7.2. Mode de joc dinàmic

En aquest mode, les cartes intercanviaran de posició quan l'usuari falli dos vegades la seva jugada. Per tant, com que aquest fet es controla mitjançant scripts, no hi han canvis als elements de l'editor gràfic respecte al mode normal, simplement s'ha d'actualitzar el nom de l'escena a l'script corresponent assignat a la càmera principal per aplicar la normativa del mode de joc corresponent.

6.7.3. Mode de joc de so

En aquest mode, el canvi principal respecte als modes anteriors és que només hi ha un únic escenari possible. Per tant, només hi haurà un fons assignat i un generador de cartes, on aquestes tindran diferents arxius de so, ja que la diferència entre cartes no serà visual sinó auditiva.

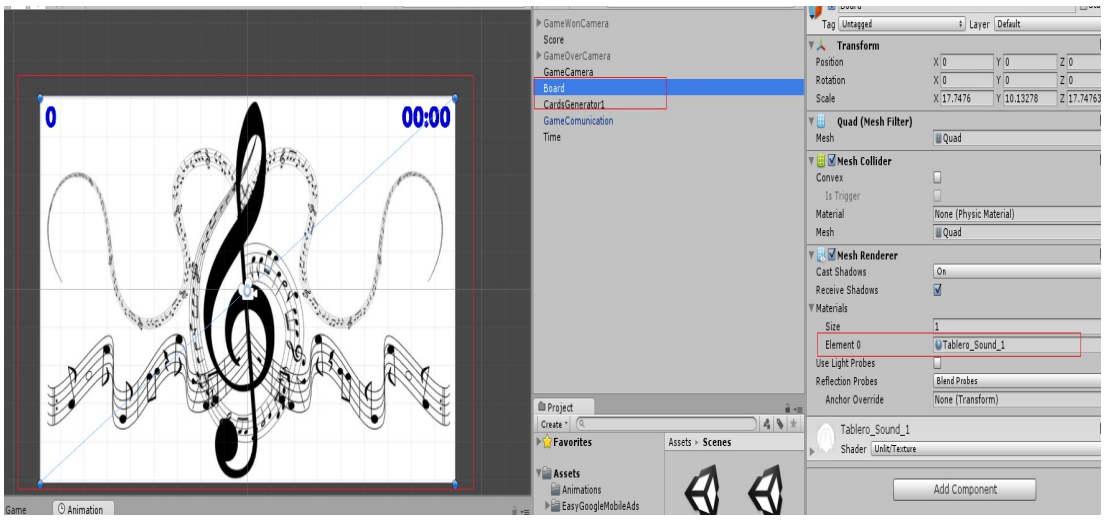


Figura 51: Diferències del mode de so.

6.8. Exportant el projecte a Android

Un cop completat el projecte, cal exportar-ho a la plataforma on volem publicar el joc. En aquest cas, volem publicar l'aplicació en la plataforma Google Play per tal que tots els usuaris que disposin d'un dispositiu amb sistema operatiu Android puguin descarregar l'aplicació i jugar. Per tant, el primer pas és realitzar la configuració necessària per poder exportar l'aplicació correctament.

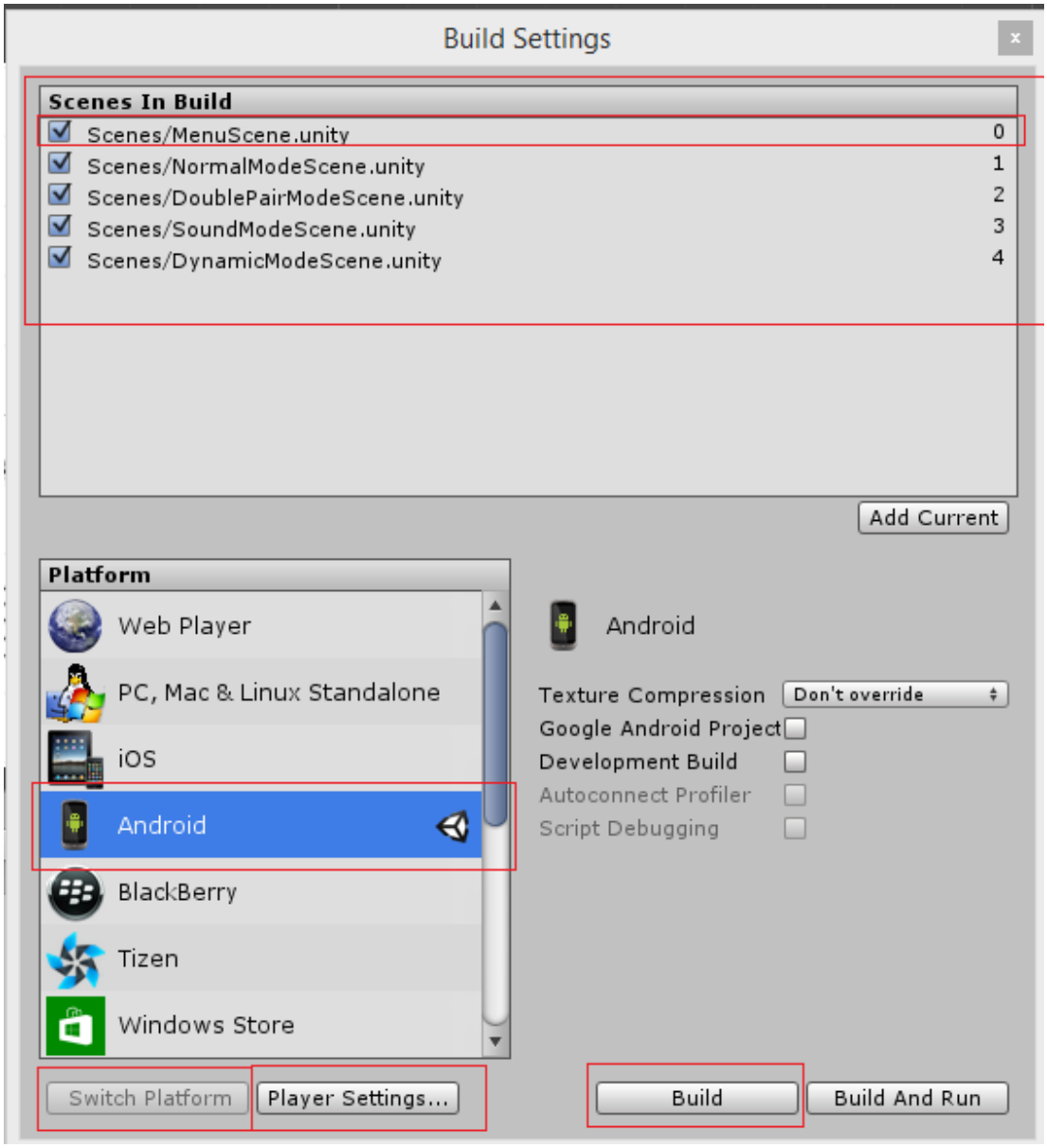


Figura 52: Exportació de l'aplicació a Android.

Primer de tot cal incloure, en la finestra *Scenes In Build*, totes les escenes del joc, on la primera del llistat serà la primera que es mostri a l'usuari. Seguidament, cal indicar quina serà la plataforma seleccionada amb el botó *Switch Platform*.

El pas més important a l'hora d'exportar l'aplicació és la configuració *Player Settings*, que veurem amb més detall posteriorment. Un cop realitzada la configuració corresponent a l'aplicació fem clic al botó *Build* i obtindrem, entre altres, l'arxiu instal·lador de l'aplicació en format *apk*.

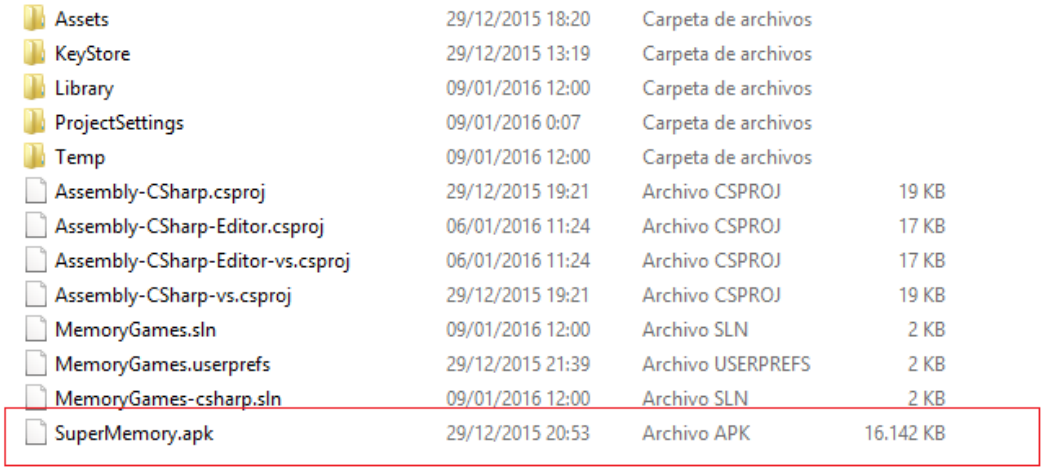


Figura 53: Arxiu instal·lador de l'aplicació.

Pel que fa la configuració corresponent a l'aplicació, només cal canviar i assignar certs camps, mentre que la resta els deixem amb els valors per defecte. Els més importants són:

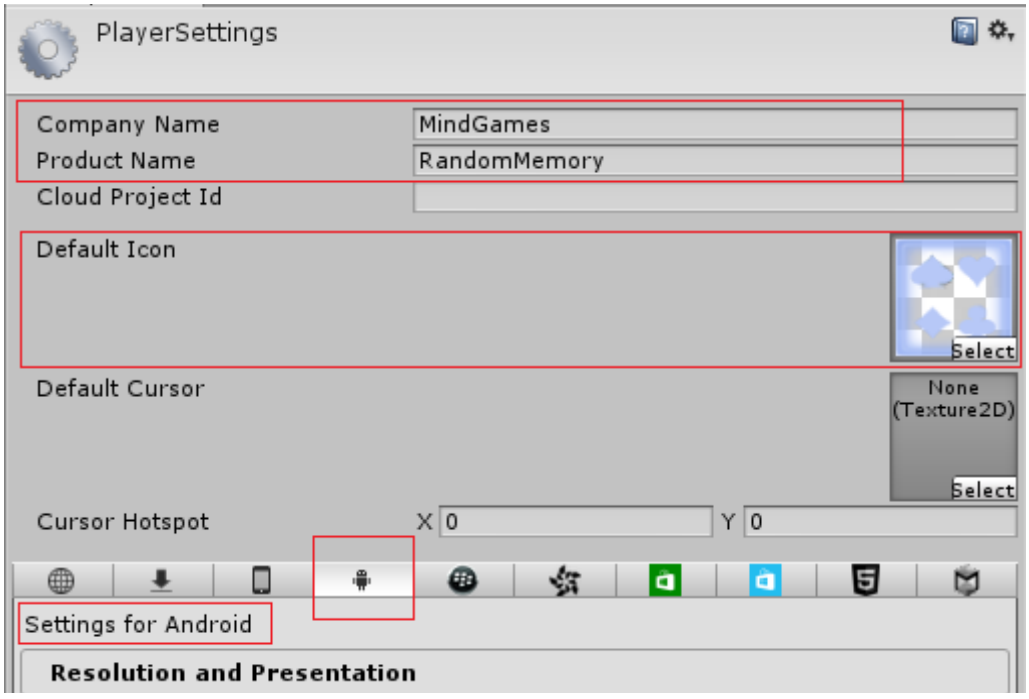


Figura 54: Configuració de l'aplicació.

- Assignar el nom de la companyia que ha creat l'aplicació.
- Assignar el nom del producte que es publicarà.
- Afegir l'icone que representarà l'aplicació en els dispositius Android.
- Seleccionar la pestanya amb l'icone del sistema Android per afegir les configuracions pròpies per el sistema seleccionat.

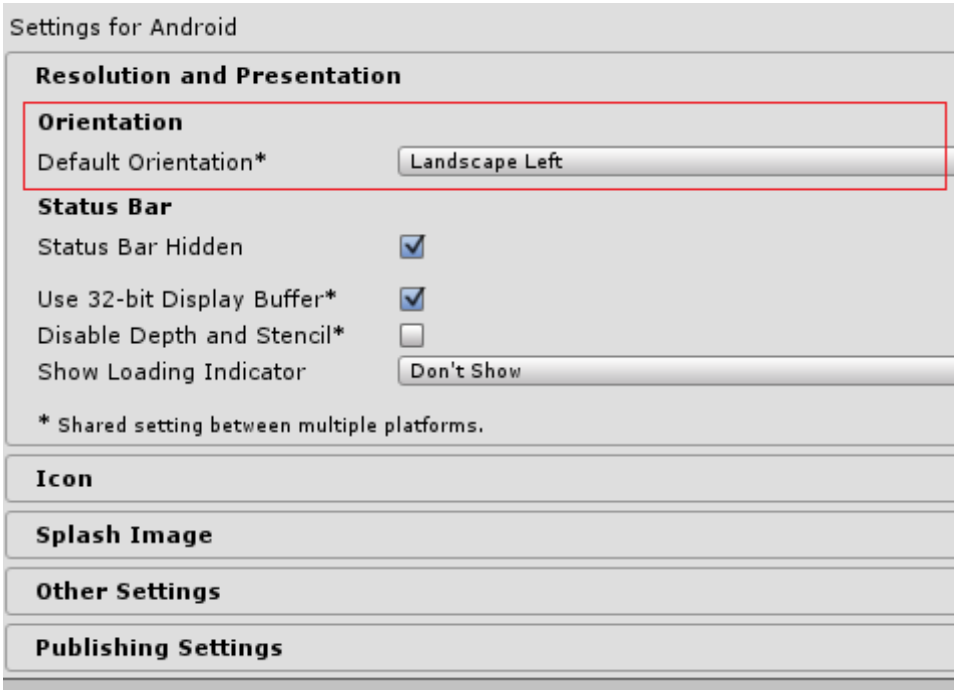


Figura 55: Resolució i presentació de l'aplicació.

En aquesta pestanya, simplement especifiquem que el contingut de l'aplicació es mostrarà en l'orientació *Landscape*. Per tant, l'aplicació es mostrarà de forma panoràmica.

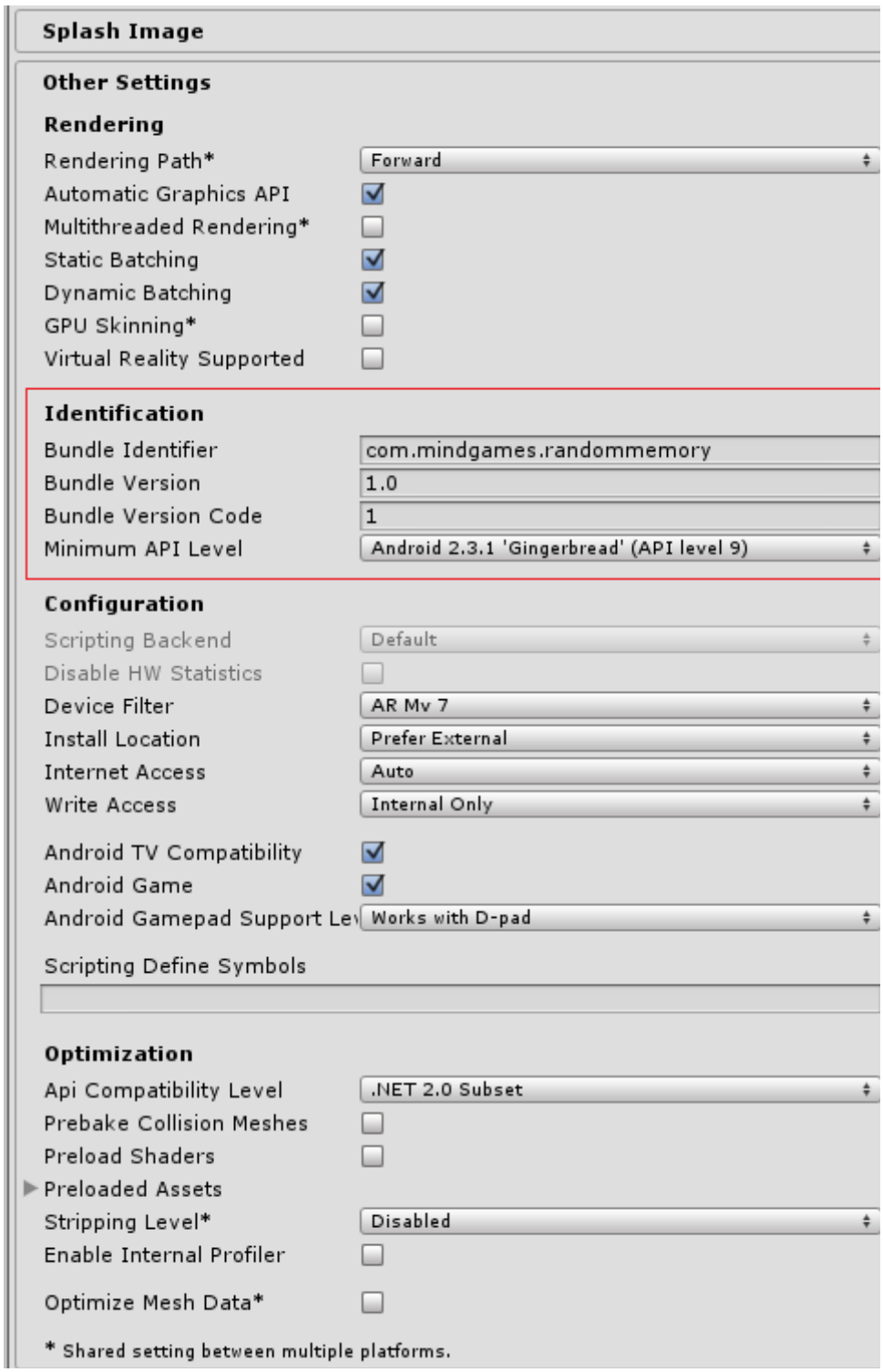


Figura 56: Altres configuracions.

Aqui determinem l'identificador del paquet de l'aplicació, que ha de contenir el nom de l'empresa i de l'aplicació exactament igual com s'ha definit anteriorment però en minúscules. Aquest identificador ha de ser únic al mercat. També cal assignar la versió de l'aplicació, que en cas d'actualitzacions cal augmentar el seu valor. Finalment, cal determinar quin sistema operatiu mínim requereix l'aplicació per ser executada.

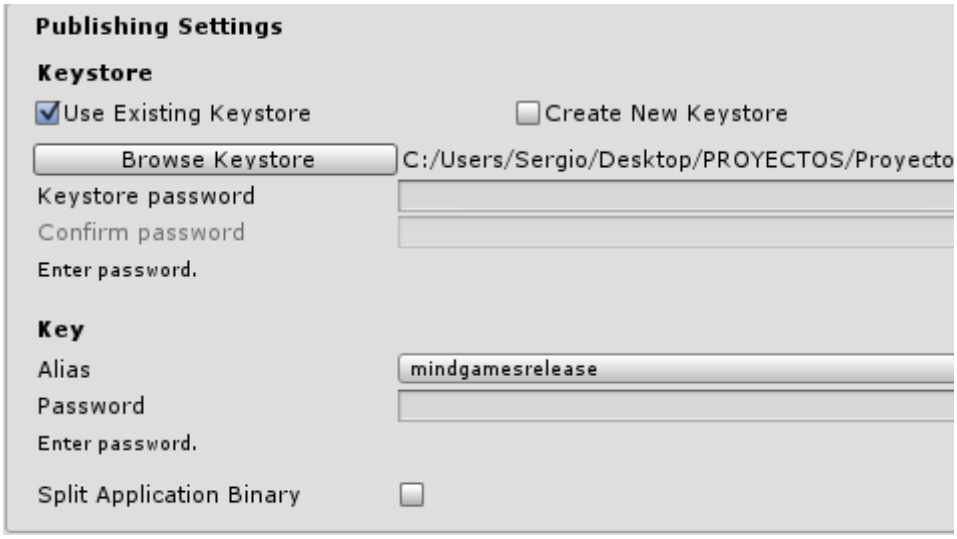


Figura 57: Firma de l'aplicació.

Aquesta última configuració és la més important de totes. Es tracta d'aplicar una firma digital al projecte per seguretat. Aquesta firma determina que l'aplicació ha estat feta per l'empresa o desenvolupador concret i, en cas de realitzar actualitzacions a l'aplicació, només es poden realitzar amb la mateixa firma.

Per tant, cal crear un fitxer que contingui les claus de l'usuari, amb contrasenyes per poder duu a terme la firma. Cal seleccionar el fitxer i escriure la contrasenya.

Un cop realitzada la configuració i la compilació, ja tenim l'aplicació llesta per realitzar la publicació a la botiga.

6.9. Monetitzant l'aplicació

Per tal de guanyar diners amb el projecte, cal afegir publicitat de AdMob per monetitzar l'aplicació. En aquest projecte afegim publicitat en forma *Intersticial* quan l'usuari prem el botó jugar del menú principal.

Primerament, cal afegir una nova aplicació al compte de AdMob. Es poden afegir de diferents formes, depenent si l'aplicació ja està publicada o no. Un cop afegida, es veurà una finestra com aquesta:

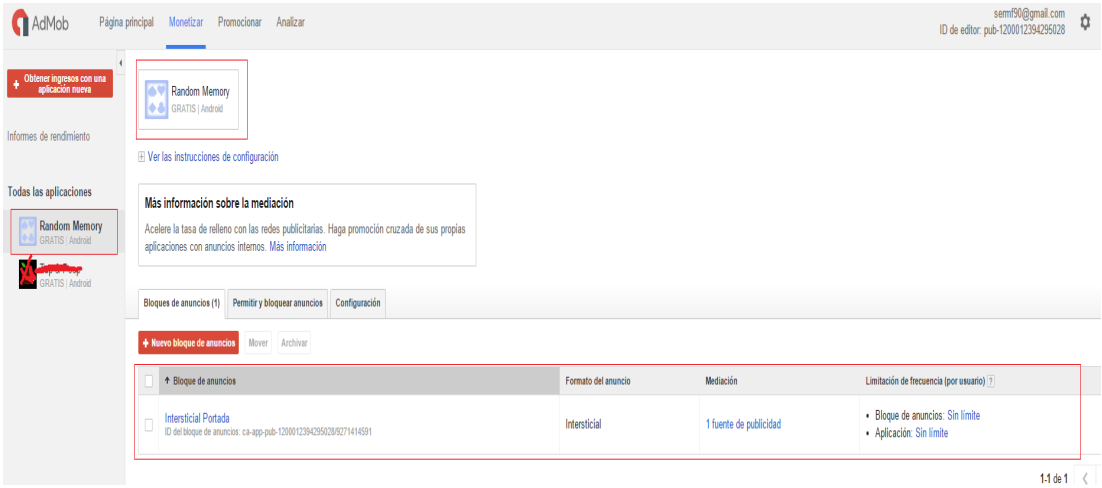


Figura 58: Creació d'anuncis a l'aplicació.

En la figura anterior podem veure que hem creat una nova publicitat i està assignada a l'aplicació. Cada una d'elles té un identificador que cal assignar als scripts, com es podrà veure més endavant. Per crear el bloc d'anuncis cal:

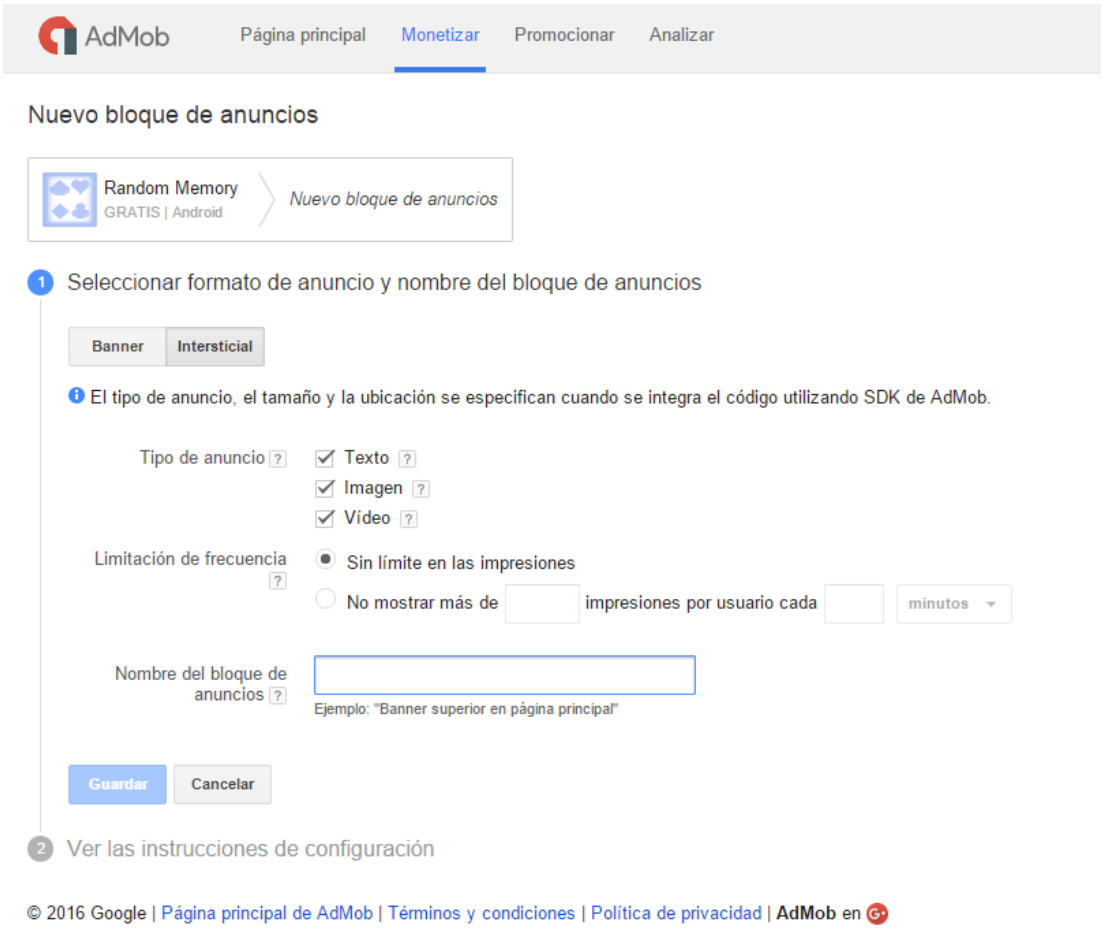


Figura 59: Creació d'un nou bloc d'anuncis.

Determinem les característiques que ens interessen segons l'aplicació i fem clic a guardar. Així, es veurà el nou bloc definit com a la figura 58, amb el seu identificador corresponent.

Per altra banda, cal importar a Unity els *plugins* necessaris per duu a terme la connexió entre la plataforma AdMob i el projecte. Per importar un paquet només cal:

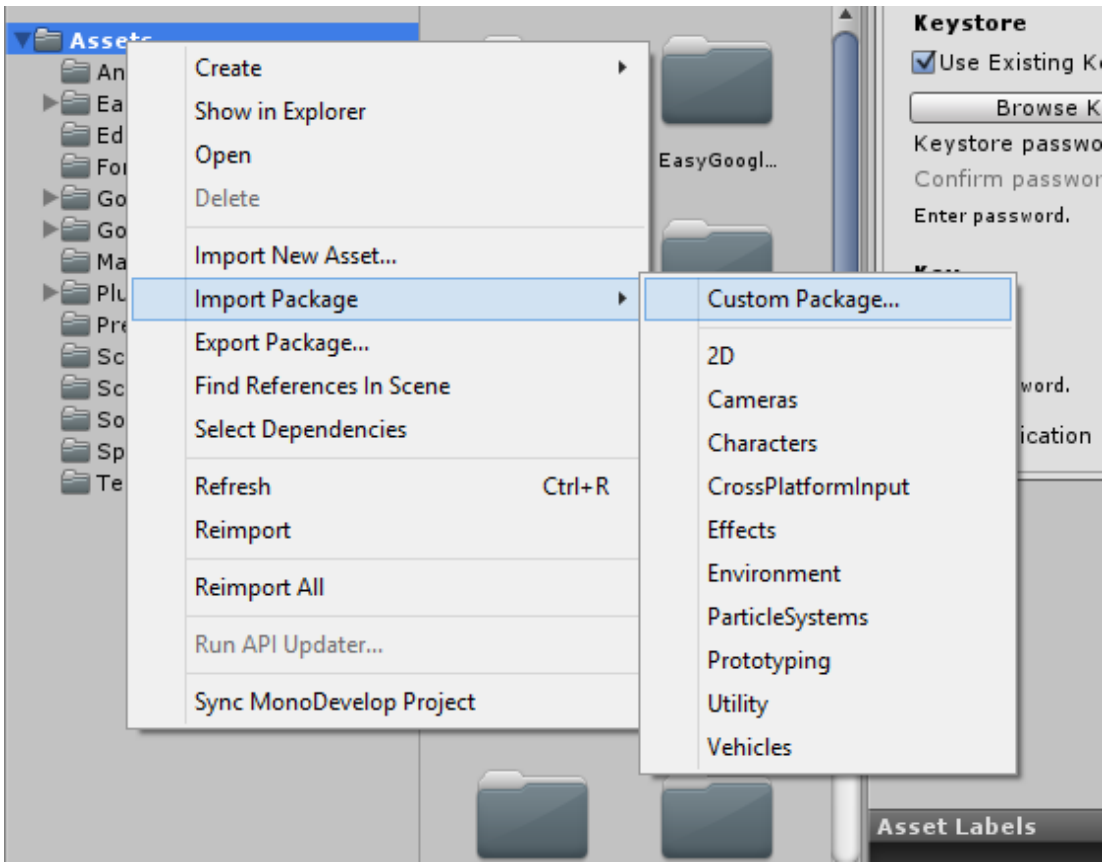


Figura 60: Importar paquets al projecte.

Seleccionem els paquets necessaris per visualitzar la publicitat a l'aplicació i el resultat serà una jerarquia de carpetes dins del projecte:

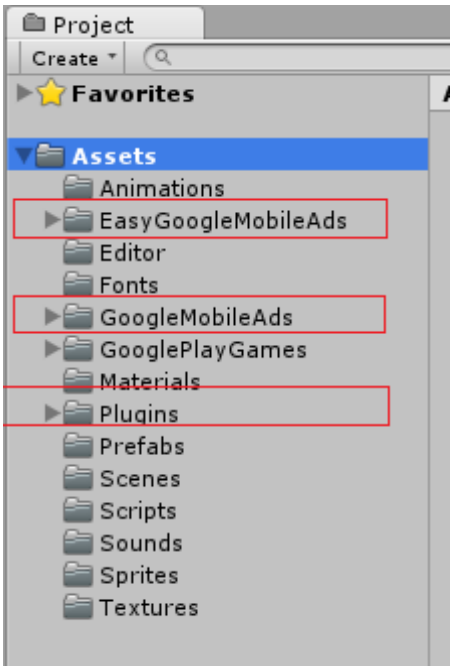


Figura 60: *Plugins* de AdMob.

Un cop realitzats aquests passos i desenvolupant la funcionalitat en els scripts, podem incloure publicitat a l'aplicació.

6.10. Google Play

Com que aquest document es basa en la creació de l'aplicació, no ens centrarem en el funcionament de la publicació d'una aplicació en Google Play. Per tant, descriurem els assoliments i classificacions creats per el nostre joc.

Els assoliments que l'usuari pot aconseguir en el nostre joc són:

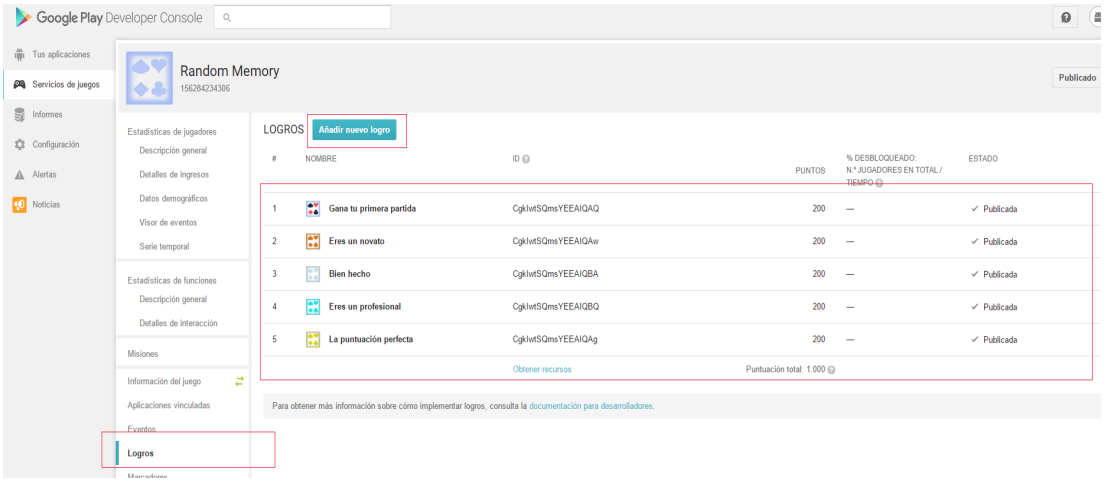


Figura 61: Assoliments en Google Play Games.

Per afegir un nou assoliment cal fer clic al botó enmarcat i seleccionar les característiques segons l'assoliment que es vol realitzar. Per exemple:

Figura 62: Assoliment per l'aplicació.

On també comptem amb l'identificador corresponent que ens identificarà l'assoliment aconseguit per l'usuari des dels scripts del projecte.

Pel que fa a la classificació d'usuaris, el sistema de creació és similar al dels assoliments.

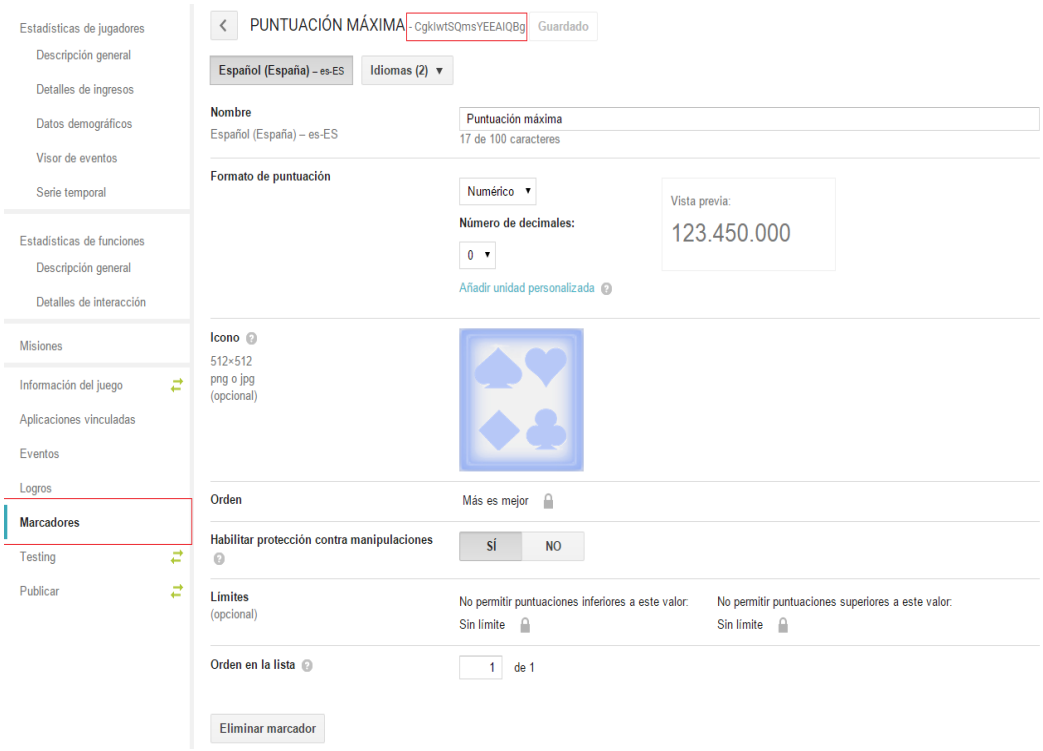


Figura 63: Classificació d'usuaris per l'aplicació.

Un pas important a l'hora de realitzar les proves en un dispositiu real abans de la publicació és assignar un compte de google per tal de no generar beneficis falsos i que Google ens tanqui el compte de desenvolupador.

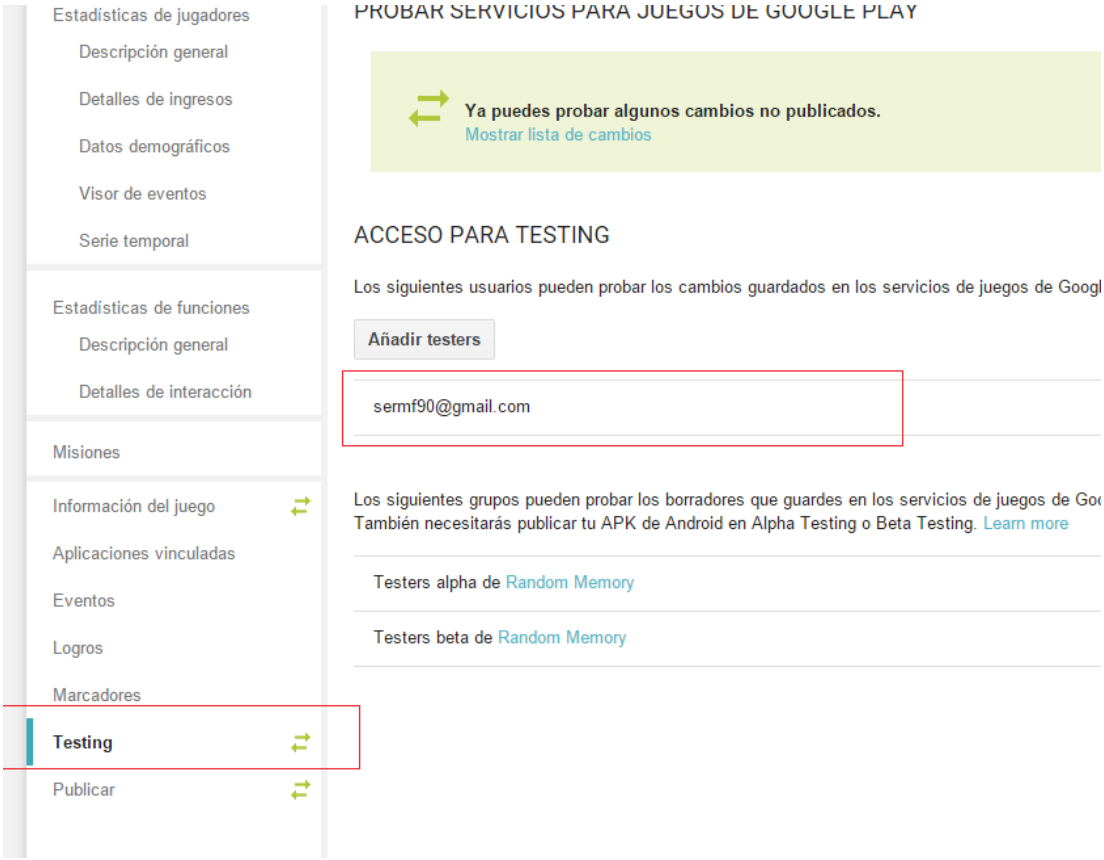


Figura 64: Especificació d'usuaris *testers*.

Per altra banda cal importar la funcionalitat necessària a Unity per poder aplicar les funcions socials a l'aplicació. Com en el cas de la publicitat, importem els paquets i es creen noves jerarquies de carpetes:

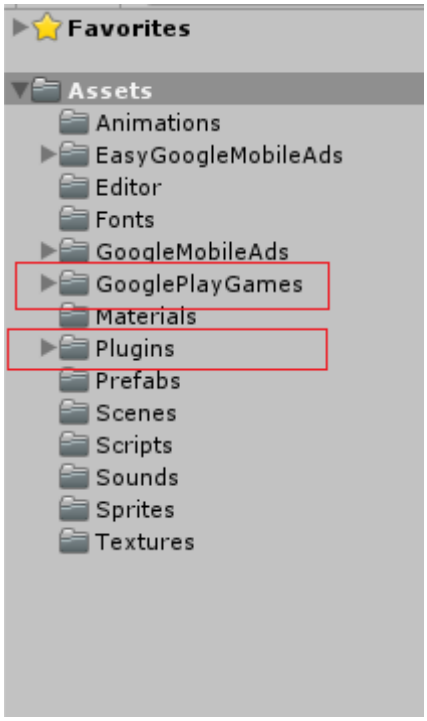


Figura 65: Importació dels *plugins* de Google Play Games.

Un cop importats els paquets i compilada l'aplicació, ens sortiran noves opcions al menú de Unity, entre elles la possibilitat d'afegir l'identificador del dispositiu on realitzarem les proves del joc. Aquest pas és important per no generar falsos ingressos a Google.

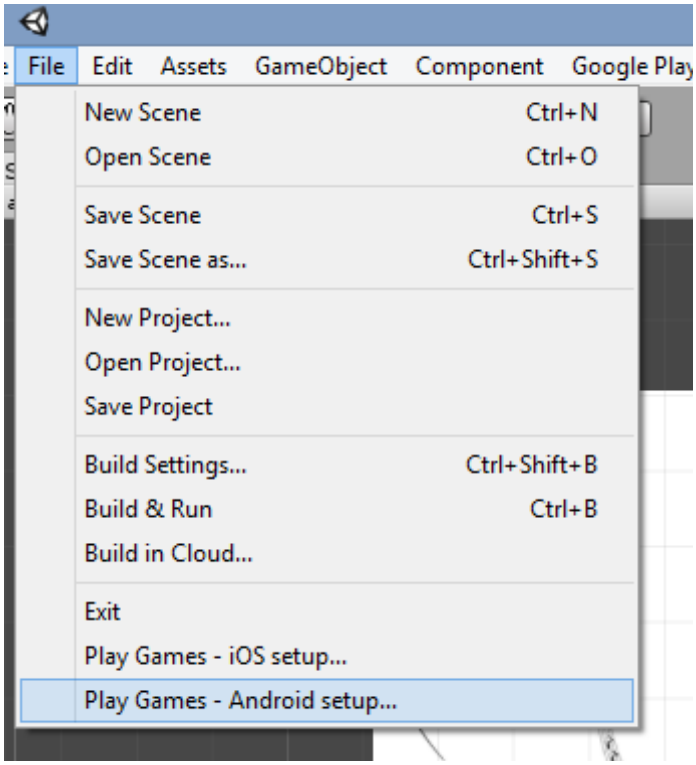
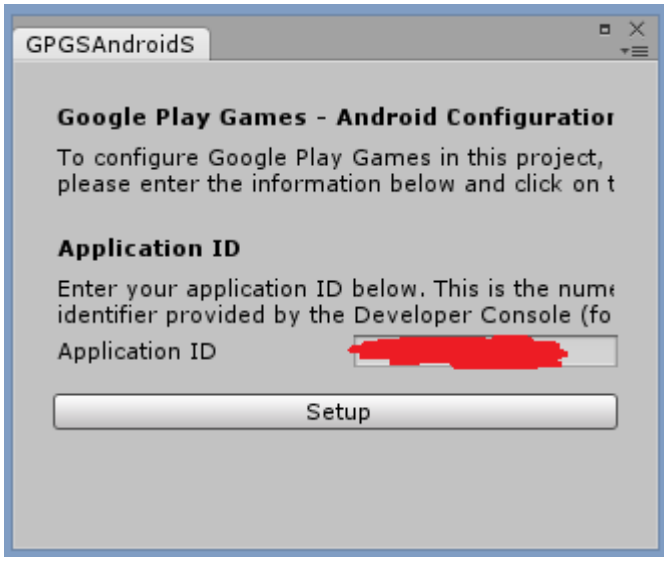


Figura 66: Noves funcionalitats de Google Play.

Per saber l'identificador del dispositiu, existeixen diverses aplicacions gratuïtes a la botiga que t'indiquen quin és l'identificador del dispositiu. Un cop el sabem, l'assignem a Unity.



Finalment, seguint el document i cercant tutorials sobre l'eina de Unity3D i sobre el funcionament de plataformes com les de Google Play i AdMob, podem obtenir *Random Memory*, una nova aplicació en la botiga de Google Play.

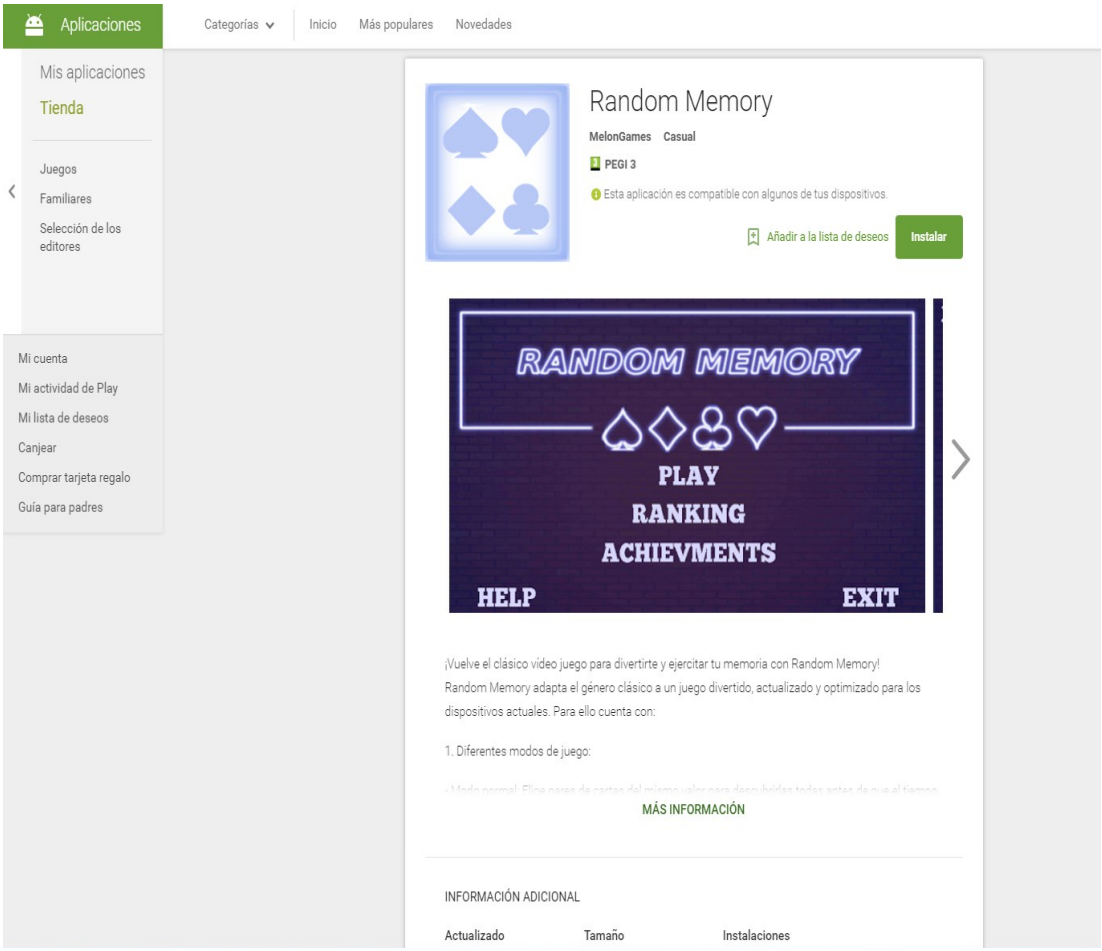


Figura 67: *Random Memory* disponible en Google Play.

7. Scripts

MonoDevelop és l'entorn de desenvolupament de codi integrat a Unity3D, amb el qual hem desenvolupat tots i cadascun dels scripts en C#.

En els següents apartats, comentarem els scripts que composen la funcionalitat del joc juntament amb el codi corresponent. Només s'explicaran les parts més importants.

7.1. AchievmentButton

Aquest script está associat a l'element que representa el botó d'assoliments del menú principal i s'encarrega principalment de mostrar els assoliments aconseguits per l'usuari des de la plataforma de Google Play Games.

En la primera part del codi declarem el component TextMesh de l'objecte al que està associat l'script. En el constructor l'obtenim de l'editor gràfic de la manera següent:

```
private TextMesh textButton;

void Awake(){
    textButton = GetComponent<TextMesh>();
}
```

Seguidament, en la funció update, volem que el color del text canviï segons l'estat de l'usuari. Si ha ingresat al seu compte de Google correctament el color del text del botó serà verd, si encara no ho ha fet, serà gris:

```
// Update is called once per frame
void Update () {
    textButton.color = Social.localUser.authenticated ? Color.green : Color.gray;
}
```

Finalment, quan l'usuari toca el botó, primerament reproduïm el so qué té com a component l'objecte. Seguidament, si l'usuari ha ingresat prèviament mostrarem els assoliments. En cas contrari, li donarem l'opció a ingresar:

```
void OnMouseDown(){
    GetComponent<AudioSource>().Play();

    if(Social.localUser.authenticated){
        Social.Active.ShowAchievementsUI();
    }else{
        Social.localUser.Authenticate((bool success) => {});
    }
}
```

Aquí es pot veure el codi complet de l'arxiu:

```
using UnityEngine;
using System.Collections;
using GooglePlayGames;
using UnityEngine.SocialPlatforms;

public class AchievmentButton : MonoBehaviour {

    private TextMesh textButton;

    void Awake(){
        textButton = GetComponent<TextMesh>();
    }

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {
        textButton.color = Social.localUser.authenticated ? Color.green : Color.gray;
    }

    void OnMouseDown(){
        GetComponent<AudioSource>().Play();

        if(Social.localUser.authenticated){
            Social.Active.ShowAchievementsUI();
        }else{
            Social.localUser.Authenticate((bool success) => {});
        }
    }
}
```

7.2. ActivateGameOver

Aquest script està associat a l'element que representa la càmera de derrota de qualsevol mode de joc. La seva funcionalitat principal és reproduïr el so corresponent i activar la càmera.

Per tant, primerament definim dos variables públiques, una en la que assignem,

des de l'editor gràfic, l'objecte de la càmera i l'altre on assignem l'arxiu de so a reproduïr:

```
//Variable que almacena la camara que se activara, indicada en el editor
public GameObject gameOverCamera;
//Variable que almacena el audio que se activara, indicado en el editor
public AudioClip gameOverClip;
```

Seguidament, afegim un observador perquè estigui atent a l'esdeveniment "GameOver" que es cridarà des d'un altre script:

```
// Use this for initialization
void Start () {
    NotificationCenter.DefaultCenter ().AddObserver (this, "GameOver");
}
```

Finalment, un cop s'ha notificat aquest esdeveniment, especifiquem que la música actual s'aturi, assignem el nou arxiu de so, indiquem que només es reprodueixi una vegada i activem la càmera:

```
void GameOver(Notification notificacion){
    //Si la partida finaliza paramos los sonidos existentes y reproducimo
    GetComponent<AudioSource>().Stop();
    GetComponent<AudioSource>().clip = gameOverClip;
    GetComponent<AudioSource>().loop = false;
    GetComponent<AudioSource>().Play();
    //Activamos la camara indicada en el editor
    gameOverCamera.SetActive(true);
}
```

Aquí es pot veure el codi complet de l'arxiu:

```
using UnityEngine;
using System.Collections;

public class ActivateGameOver : MonoBehaviour {

    //Variable que almacena la camara que se activara, indicada en el editor
    public GameObject gameOverCamera;
    //Variable que almacena el audio que se activara, indicado en el editor
    public AudioClip gameOverClip;

    // Use this for initialization
    void Start () {
        NotificationCenter.DefaultCenter ().AddObserver (this, "GameOver");
    }

    void GameOver(Notification notificacion){
        //Si la partida finaliza paramos los sonidos existentes y reproducimos el asignado
        GetComponent<AudioSource>().Stop();
        GetComponent<AudioSource>().clip = gameOverClip;
        GetComponent<AudioSource>().loop = false;
        GetComponent<AudioSource>().Play();
        //Activamos la camara indicada en el editor
        gameOverCamera.SetActive(true);
    }

    // Update is called once per frame
    void Update () {

    }
}
```

7.3. ActivateGameWon

Aquest script està associat a l'element que representa la càmera de victòria de qualsevol mode de joc. La seva funcionalitat principal és reproduïr el so corresponent i activar la càmera. Per tant, la funcionalitat és exactament la mateixa exceptuant que disposa d'un altre notificador diferent a l'anterior:

```
// Use this for initialization
void Start () {
    NotificationCenter.DefaultCenter ().AddObserver (this, "GameResults");
}
```

Aquí es pot veure el codi complet de l'arxiu:

```
using UnityEngine;
using System.Collections;

public class ActivateGameWon : MonoBehaviour {

    //Variable que almacena la camara que se activara, indicada en el editor
    public GameObject gameWonCamera;
    //Variable que almacena el audio que se activara, indicado en el editor
    public AudioClip gameWonClip;

    // Use this for initialization
    void Start () {
        NotificationCenter.DefaultCenter ().AddObserver (this, "GameResults");
    }

    void GameResults(Notification notificacion){
        //Si la partida finaliza paramos los sonidos existentes y reproducimos el asignado
        GetComponent<AudioSource>().Stop();
        GetComponent<AudioSource>().clip = gameWonClip;
        GetComponent<AudioSource>().loop = false;
        GetComponent<AudioSource>().Play();
        //Activamos la camara indicada en el editor
        gameWonCamera.SetActive(true);
    }

    // Update is called once per frame
    void Update () {

    }
}
```

7.4. AspectUtility

Aquest script està associat a totes les càmeres de qualsevol mode de joc. La seva funcionalitat principal és ajustar la relació d'aspecte del joc segons la pantalla.

En aquest cas, la part principal de l'script està en la següent funció:

```
public static void SetCamera () {
    float currentAspectRatio = (float)Screen.width / Screen.height;

    if ((int)(currentAspectRatio * 100) / 100.0f == (int)(wantedAspectRatio * 100) / 100.0f) {
        cam.rect = new Rect(0.0f, 0.0f, 1.0f, 1.0f);
        if (backgroundCam) {
            Destroy(backgroundCam.gameObject);
        }
        return;
    }
    // Pillarbox
    if (currentAspectRatio > wantedAspectRatio) {
        float inset = 1.0f - wantedAspectRatio/currentAspectRatio;
        cam.rect = new Rect(inset/2, 0.0f, 1.0f-inset, 1.0f);
    }
    // Letterbox
    else {
        float inset = 1.0f - currentAspectRatio/wantedAspectRatio;
        cam.rect = new Rect(0.0f, inset/2, 1.0f, 1.0f-inset);
    }
    if (!backgroundCam) {
        backgroundCam = new GameObject("BackgroundCam", typeof(Camera)).GetComponent<Camera>();
        backgroundCam.depth = int.MinValue;
        backgroundCam.clearFlags = CameraClearFlags.SolidColor;
        backgroundCam.backgroundColor = Color.black;
        backgroundCam.cullingMask = 0;
    }
}
```

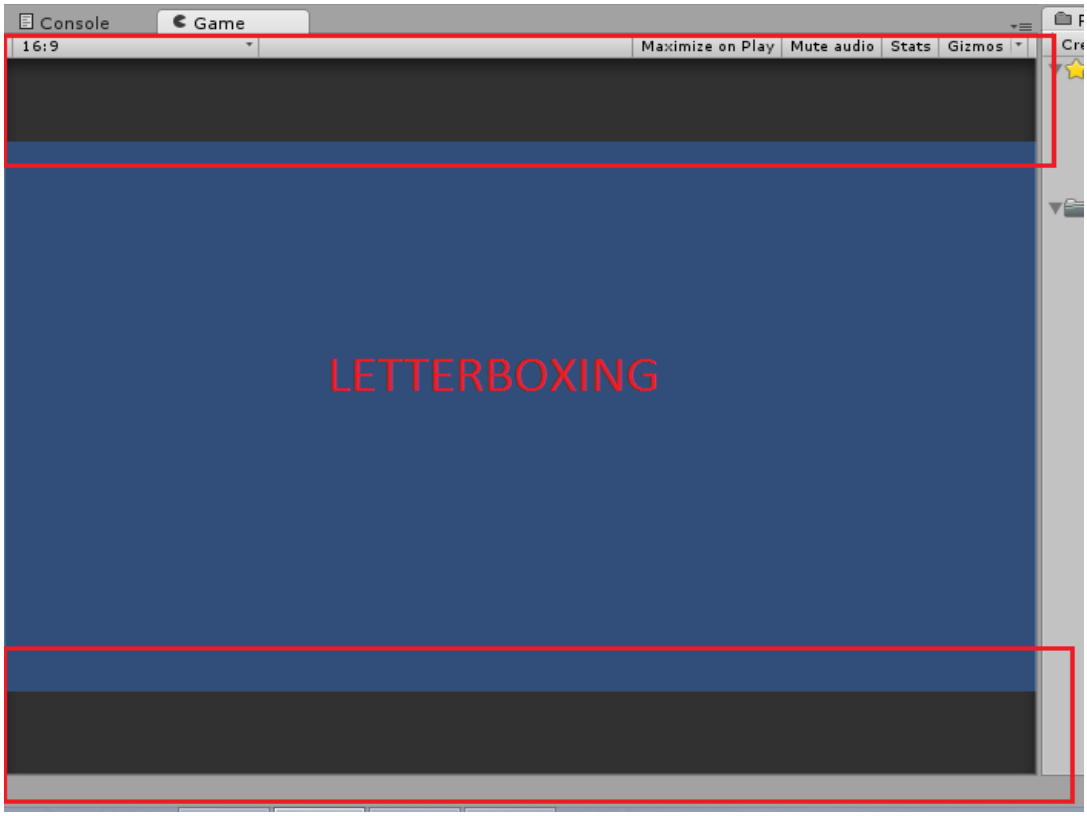
Abans de continuar, cal introduir els dos conceptes nous que han aparegut. Què és el *letterboxing* i el *pillarboxing*?

Letterboxing / Pillarboxing

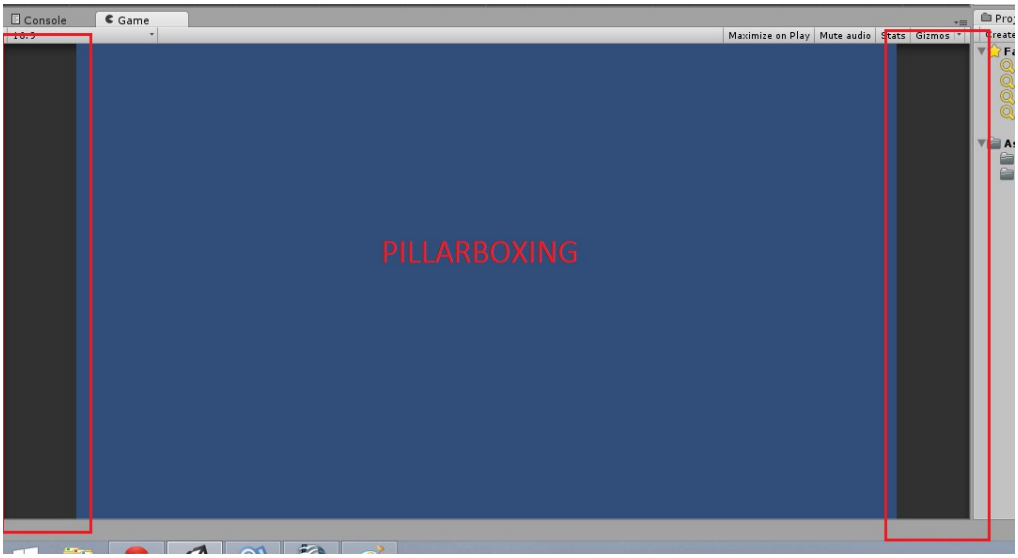
Tan *letterboxing* com *pillarboxing* són pràctiques d'adaptar un element visual amb una relació d'aspecte concreta a una pantalla final amb una relació diferent. Llavors, com que queda espai a la pantalla sense definir cal aplicar aquestes tècniques, amb un fons negre, segons sigui la pantalla.

Per tant, segons la relació d'aspecte de la pantalla cal afegir unes barres horitzontals o verticals que simulen un fons negre, que s'adapten a la pantalla de manera que no es veu cap espai sense definir.

Aleshores, cal aplicar *letterboxing* quan la pantalla del dispositiu té massa alçada per la relació d'aspecte de l'aplicació:



Per conseqüència, en cas que la **pantalla del dispositiu tingui massa amplada** per la relació d'aspecte de l'aplicació cal aplicar **pillarboxing**:



Un cop entesos aquests dos conceptes passem a veure la funcionalitat específica de l'script que fa possible això.

Primerament calculem la relació d'aspecte de la pantalla on es carrega l'aplicació, dividint l'amplada d'aquesta per l'alçada. Si aquesta relació d'aspecte és la mateixa que la que hem assignat a la càmera des de l'editor gràfic, crearem el rectangle de la càmera normal i destruirem la càmera de fons negre.

En canvi, si la relació d'aspecte és diferent a l'assignada, cal comprovar on varia i crear els respectius rectangles per saber si cal aplicar *Pillarbox* o *Letterbox*, ja esmentats anteriorment. Fora d'aquesta funció, s'han creat mètodes adicionals a cridar en cas d'utilitzar els conceptes anteriors.

Finalment, si la càmera de fons no existeix, la creem amb les característiques corresponents. Aquí es pot veure la resta de codi de l'arxiu:

```
using UnityEngine;

public class AspectUtility : MonoBehaviour {

    //Variable publica en el que asignamos el valor mediante el editor grafico
    public float _wantedAspectRatio = 1.3333333f;
    //Variable estatica donde guardamos el valor anterior
    static float wantedAspectRatio;
    //Objeto de la camara asignada
    static Camera cam;
    //Objeto de la camara trasera para fondo negro
    static Camera backgroundCam;


    /* Constructor
     * - Recupera el componente de la camara y sino asigna la camara principal
     * - En caso de no existir ninguna camara muestra el error en el log
     * - Finalmente, asignamos el valor de la relacion de aspecto que indicamos en el editor grafico
     * a la variable estatica wantedAspectRatio
     * - Iniciamos la funcion SetCamera()*/
    void Awake () {
        cam = GetComponent<Camera>();
        if (!cam) {
            cam = Camera.main;
        }
        if (!cam) {
            Debug.LogError ("No camera available");
            return;
        }
        wantedAspectRatio = _wantedAspectRatio;
        SetCamera();
    }
```

```
/* Configuracion de la camara
 * - Guardamos la relacion de aspecto de la pantalla del dispositivo en la variable currentAspectRatio
 * - En caso de ser igual al valor asignado previamente creamos el rectangulo a pantalla completa y
 *  y eliminamos la camara trasera si hubiera alguna (camara para añadir fondo negro)
 * - En caso de que la relacion de aspecto del dispositivo sea mayor que la asignada creamos un pillarbox
 * - En otro caso, creamos una letterbox
 * - Finalmente, en caso de no existir camara trasera creamos una con el fondo de color negro. Esto
 *  es para que en casos de que la relacion de aspecto no se ajuste a la pantalla del dispositivo
 *  la camara trasera actue como un fondo negro (pillarboxing/letterboxing).
 */
public static void SetCamera () {
    float currentAspectRatio = (float)Screen.width / Screen.height;

    if ((int)(currentAspectRatio * 100) / 100.0f == (int)(wantedAspectRatio * 100) / 100.0f) {
        cam.rect = new Rect(0.0f, 0.0f, 1.0f, 1.0f);
        if (backgroundCam) {
            Destroy(backgroundCam.gameObject);
        }
        return;
    }
    // Pillarbox
    if (currentAspectRatio > wantedAspectRatio) {
        float inset = 1.0f - wantedAspectRatio/currentAspectRatio;
        cam.rect = new Rect(inset/2, 0.0f, 1.0f-inset, 1.0f);
    }
    // Letterbox
    else {
        float inset = 1.0f - currentAspectRatio/wantedAspectRatio;
        cam.rect = new Rect(0.0f, inset/2, 1.0f, 1.0f-inset);
    }
    if (!backgroundCam) {
   public static Vector2 guiMousePosition {
       get {
           Vector2 mousePos = Event.current.mousePosition;
           mousePos.y = Mathf.Clamp(mousePos.y, cam.rect.y * Screen.height, cam.rect.y * Screen.height + cam.rect.height * Screen.height);
           mousePos.x = Mathf.Clamp(mousePos.x, cam.rect.x * Screen.width, cam.rect.x * Screen.width + cam.rect.width * Screen.width);
           return mousePos;
       }
   }

   //-------------------------------------------------------------------------------------------------------------------
}
```

```
//------------Funciones utilizadas en caso de usar Pillarbox o Letterbox (ya que entonces algunos de los calculos
//            anteriores no seran correctos, ya que se calculara el tamaño de la pantalla y no el de la camara)-------------------------
public static int screenHeight {
    get {
        return (int)(Screen.height * cam.rect.height);
    }
}

public static int screenWidth {
    get {
        return (int)(Screen.width * cam.rect.width);
    }
}

public static int xOffset {
    get {
        return (int)(Screen.width * cam.rect.x);
    }
}

public static int yOffset {
    get {
        return (int)(Screen.height * cam.rect.y);
    }
}

public static Rect screenRect {
    get {
        return new Rect(cam.rect.x * Screen.width, cam.rect.y * Screen.height, cam.rect.width * Screen.width, cam.rect.height * Screen.height);
    }
}

public static Vector3 mousePosition {
    get {
        Vector3 mousePos = Input.mousePosition;
        mousePos.y -= (int)(cam.rect.y * Screen.height);
        mousePos.x -= (int)(cam.rect.x * Screen.width);
        return mousePos;
    }
}
```

7.5. BoardGenerator

Aquest script està associat al generador de fons de partida i la seva funcionalitat principal és generar un tauler aleatori i activar el generador de les cartes, marcador i temps corresponent al fons seleccionat.

Primerament, definim les variables seguents:

```
//Guardamos los escenarios posibles para mostrar
public Material[] backgrounds;
//Guardamos los generadores de cartas correspondientes a los escenarios
public GameObject[] generators;
//Guardamos el marcador y el cronómetro para cambiarles el color del texto
public TextMesh timer;
public TextMesh score;
//Guardamos una serie de colores para asignar según el escenario
private Color[] colorCodes;
```

- Assignem, des de l'editor gràfic, els materials que possiblement formaran part del component *Renderer* del fons de pantalla.

- Assignem els objectes dels generadors de cartes existents correponents als escenaris.

- Assignem, també, els components *TextMesh* corresponents del marcador i del temps.

- Finalment, creem una variable on es desaran els colors que, posteriorment, s'assignaran als components anteriors.

Seguidament, en la funció start, mitjançant la funcionalitat que ens ofereix la clase Random, recuperem un valor aleatori entre el 0 i la longitut de la cadena de materials. A partir d'aquí, seleccionem i assignem les característiques corresponents al material seleccionat de la manera seguent:

```
// Use this for initialization
void Start () {
    //Variable para determinar el numero aleatorio para crear el escenario
    int random = Random.Range(0,backgrounds.Length);
    //Inicializamos los códigos de los colores
    this.colorCodes = new Color[backgrounds.Length];
    this.colorCodes[0] = new Color (0.84f,0.84f,1.00f,1.00f);
    this.colorCodes[1] = new Color (0.09f,0.73f,0.04f,1.00f);
    this.colorCodes[2] = new Color (0.89f,0.87f,0.16f,1.00f);
    this.colorCodes [3] = Color.white;


    //Asignamos el fondo de pantalla
    this.gameObject.GetComponent<Renderer> ().material = backgrounds [random];
    //Asignamos el color del texto de la pantalla
    this.timer.color = this.colorCodes[random];
    this.score.color = this.colorCodes[random];
    //Dependiendo del escenario elegido, activar el generador correspondiente.
    generators [random].SetActive (true);

}
```

Aquí podem veure el codi complet de l'arxiu:

```
using UnityEngine;
using System.Collections;

public class BoardGenerator : MonoBehaviour {

    //Guardamos los escenarios posibles para mostrar
    public Material[] backgrounds;
    //Guardamos los generadores de cartas correspondientes a los escenarios
    public GameObject[] generators;
    //Guardamos el marcador y el cronómetro para cambiarles el color del texto
    public TextMesh timer;
    public TextMesh score;
    //Guardamos una serie de colores para asignar según el escenario
    private Color[] colorCodes;

    // Use this for initialization
    void Start () {
        //Variable para determinar el numero aleatorio para crear el escenario
        int random = Random.Range(0,backgrounds.Length);
        //Inicializamos los códigos de los colores
        this.colorCodes = new Color[backgrounds.Length];
        this.colorCodes[0] = new Color (0.84f,0.84f,1.00f,1.00f);
        this.colorCodes[1] = new Color (0.09f,0.73f,0.04f,1.00f);
        this.colorCodes[2] = new Color (0.89f,0.87f,0.16f,1.00f);
        this.colorCodes [3] = Color.white;


        //Asignamos el fondo de pantalla
        this.gameObject.GetComponent<Renderer> ().material = backgrounds [random];
        //Asignamos el color del texto de la pantalla
        this.timer.color = this.colorCodes[random];
        this.score.color = this.colorCodes[random];
        //Dependiendo del escenario elegido, activar el generador correspondiente.
        generators [random].SetActive (true);

    }

    // Update is called once per frame
    void Update () {

    }
}
```

7.6. ButtonActivateCamera

Aquest script esta associat als botons del menú principal i possibilita la navegació entre les diferents pantalles del menú, activant i desactivant càmeres.

Primerament, assignem des de l'editor gràfic els objectes corresponents a les càmeres que volem activar i desactivar. Seguidament, creem una variable que ens indica si, quan l'usuari toca el botó, hem de mostrar publicitat. Aquesta indicació també es fa des de l'editor gràfic, ja que està definida com una variable pública. L'únic botó que té aquesta variable com a valor *true* és el de jugar, per no carregar l'aplicació amb publicitat i fer que l'experiència a l'usuari sigui agoviant.

```
//Nombres de las camaras que se activaran y desactivaran
public GameObject cameraToEnable;
public GameObject cameraToDisable;
public bool showIntersticial = false;
```

Respecte a la funcionalitat principal, es tracta d'un codi molt simple en el que, al tocar el botó, es reprodueix l'arxiu de so corresponent i, si en aquest botó cal mostrar publicitat, es mostra. Seguidament, invoquem una funcionalitat que activa i desactiva les càmeres assignades.

```
//Cuando el usuario toca el boton
void OnMouseDown(){
    //Reproducimos el sonido del boton
    GetComponent<AudioSource>().Play();
    if(this.showIntersticial == true) EasyGoogleMobileAds.GetInterstitialManager().ShowInterstitial();
    //Llamamos a la funcion, en 0 segundos, para activar y desactivar camaras
    Invoke("EnableCamera", 0);
}

void EnableCamera(){
    cameraToDisable.SetActive (false);
    cameraToEnable.SetActive (true);

}
```

Aquí podem veure el codi complet de l'arxiu:

```
using UnityEngine;
using System.Collections;

public class ButtonActivateCamera : MonoBehaviour {

    //Nombres de las camaras que se activaran y desactivaran
    public GameObject cameraToEnable;
    public GameObject cameraToDisable;
    public bool showIntersticial = false;


    // Use this for initialization
    void Start () {


    }

    // Update is called once per frame
    void Update () {

    }

    //Cuando el usuario toca el boton
    void OnMouseDown(){
        //Reproducimos el sonido del boton
        GetComponent<AudioSource>().Play();
        if(this.showIntersticial == true) EasyGoogleMobileAds.GetInterstitialManager().ShowInterstitial();
        //Llamamos a la funcion, en 0 segundos, para activar y desactivar camaras
        Invoke("EnableCamera", 0);
    }

    void EnableCamera(){
        cameraToDisable.SetActive (false);
        cameraToEnable.SetActive (true);

    }



}
```

7.7. ButtonLoadScene

Aquest script és similar a l'anterior. Està associat als botons de la selecció de modes i l'única responsabilitat que té és carregar l'escena corresponent al mode de joc seleccionat.

La variable pública que definim es una cadena de text en la qual, des de l'editor gràfic, especifiquem el nom de l'escena que volem carregar. Aquest nom correspon als noms dels arxius creats a la carpeta de les escenes.

```
public string nameScene = "Scene";
```

Seguidament, quan el jugador prem el botó del mode, es carrega i es mostra l'escena corresponent.

```
//Al pulsar el boton
void OnMouseDown(){

    Invoke("LoadScene", 0);
}

void LoadScene(){

    Application.LoadLevel (nameScene);
}
```

Aquí podem veure el codi complet de l'arxiu:

```
using UnityEngine;
using System.Collections;

public class ButtonLoadScene : MonoBehaviour {


    public string nameScene = "Scene";


    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }

    //Al pulsar el boton
    void OnMouseDown(){

        Invoke("LoadScene", 0);
    }

    void LoadScene(){

        Application.LoadLevel (nameScene);
    }
}
```

7.8. CardsGenerator

Aquest script està associat als generadors de cartes de les escenes i la seva funcionalitat principal és generar el conjunt de cartes aleatòriament amb les seves posicions corresponents. D'aquesta manera, a cada partida les cartes tenen posicions diferents.

En aquest cas, definim una cadena d'objectes on es desaran totes les cartes necessàries per poder completar una partida. És a dir, cada generador de cartes té un tipus de temàtica. També, definim dos variables que indiquen la separació que ha d'haver entre les cartes per ser col·locades correctament en la pantalla.

```
//Cartas seleccionadas para la partida
public GameObject[] cards;
//Separacion entre cartas
public float xSeparation;
public float ySeparation;
```

En la funció *Start* afegim tots els observadors necessaris per la creació de les cartes. Aquests corresponen a l'inici de qualsevol de les escenes dels modes. En aquest cas, en els quatre modes existents generem les cartes de manera normal però està codificat així per futures actualitzacions. Per exemple, si afegim nous modes on les cartes s'han de generar de manera diferent només caldrà realitzar canvis mínims.

```
// Use this for initialization
void Start () {
    //Añadimos un observador a este componente para que observe el evento SceneBegin,
    //que posteriormente llamara a esa funcion cuando se notifique en el Center que la escena
    //ha empezado.
    NotificationCenter.DefaultCenter().AddObserver(this, "NormalModeSceneBegin");
    NotificationCenter.DefaultCenter().AddObserver(this, "DoublePairModeSceneBegin");
    NotificationCenter.DefaultCenter().AddObserver(this, "DynamicModeSceneBegin");
    NotificationCenter.DefaultCenter().AddObserver(this, "SoundModeSceneBegin");

}
```

La funció principal és la de creació de les cartes. Seleccionem una carta aleatòriament de la cadena i la instanciem en la pantalla. Seguidament, actualitzem la posició del generador segons el número de cartes col·locades en el mapa i reorganitzem la cadena de cartes de manera que no es tornin a repetir.

```
for (int i=0; i<cards.Length; i++) {
    //Variable para determinar el numero aleatorio para crear la carta
    int random = Random.Range(0,cards.Length-countCards);
    //Creamos una carta aleatoria del listado del editor
    //Tambien indicamos la posicion del generador, ya que sino creara las cartas en el mismo sitio
    Instantiate(cards[random], transform.position, Quaternion.identity);
    //Actualizamos el numero de cartas colocadas en la fila
    numberOfCards = numberOfCards+1;
    //Comprovamos si ya hemos completado la fila
    if(numberOfCards == 8){
        //Actualizamos la posicion x del generador al inicial y actualizamos la posicion y
        transform.position = new Vector3(xFirstPosition, transform.position.y - ySeparation, transform.position.z);
        //Actualizamos el numero de cartas de la fila a 0
        numberOfCards = 0;
    }else{
        //Actualizamos la posicion x del generador
        transform.position = new Vector3(transform.position.x + xSeparation, transform.position.y, transform.position.z);
    }

    //Reorganizamos el listado de cartas para que no sea posible que se vuelva a repetir
    if(random != cards.Length-1-countCards){
        card = cards[cards.Length-1-countCards];
        cards[cards.Length-1-countCards] = cards[random];
        cards[random] = card;

    }
    countCards++;
```

Aquí es pot veure el codi complet de l'arxiu:

```
using UnityEngine;
using System.Collections;

public class CardsGenerator : MonoBehaviour {

    //Cartas seleccionadas para la partida
    public GameObject[] cards;
    //Separacion entre cartas
    public float xSeparation;
    public float ySeparation;


    // Use this for initialization
    void Start () {
        //Añadimos un observador a este componente para que observe el evento SceneBegin,
        //que posteriormente llamara a esa funcion cuando se notifique en el Center que la escena
        //ha empezado.
        NotificationCenter.DefaultCenter().AddObserver(this, "NormalModeSceneBegin");
        NotificationCenter.DefaultCenter().AddObserver(this, "DoublePairModeSceneBegin");
        NotificationCenter.DefaultCenter().AddObserver(this, "DynamicModeSceneBegin");
        NotificationCenter.DefaultCenter().AddObserver(this, "SoundModeSceneBegin");

    }

    //Esta funcion es llamada por el centro de notificaciones. Por lo tanto, requiere una notificacion como parametro
    void NormalModeSceneBegin (Notification notification) {
        //Generamos las cartas para el modo normal
        GenerateCardsNormal ();
    }

    //Esta funcion es llamada por el centro de notificaciones. Por lo tanto, requiere una notificacion como parametro
    void DoublePairModeSceneBegin (Notification notification) {
        //Generamos las cartas para el modo de doble pareja
        GenerateCardsNormal ();
    }

    //Esta funcion es llamada por el centro de notificaciones. Por lo tanto, requiere una notificacion como parametro
    void DynamicModeSceneBegin (Notification notification) {
        //Generamos las cartas para el modo dinámico
        GenerateCardsNormal ();
    }
```

```
//Esta funcion es llamada por el centro de notificaciones. Por lo tanto, requiere una notificacion como parametro
void SoundModeSceneBegin (Notification notification) {
    //Generamos las cartas para el modo de sonido
    GenerateCardsNormal ();
}

// Update is called once per frame
void Update () {

}

//Funcion para generar las cartas en tablero de 32
void GenerateCardsNormal(){

    //Posicion x inicial del generador
    float xFirstPosition = transform.position.x;
    //Cartas colocadas en la fila
    int numberOfCards = 0;
    //Variables asistentes para canviar valores de la cadena de cartas
    GameObject card;
    int countCards = 0;
```

```
        for (int i=0; i<cards.Length; i++) {
            //Variable para determinar el numero aleatorio para crear la carta
            int random = Random.Range(0,cards.Length-countCards);
            //Creamos una carta aleatoria del listado del editor
            //Tambien indicamos la posicion del generador, ya que sino creara las cartas en el mismo sitio
            Instantiate(cards[random], transform.position, Quaternion.identity);
            //Actualizamos el numero de cartas colocadas en la fila
            numberOfCards = numberOfCards+1;
            //Comprovamos si ya hemos completado la fila
            if(numberOfCards == 8){
                //Actualizamos la posicion x del generador al inicial y actualizamos la posicion y
                transform.position = new Vector3(xFirstPosition, transform.position.y - ySeparation, transform.position.z);
                //Actualizamos el numero de cartas de la fila a 0
                numberOfCards = 0;
            }else{
                //Actualizamos la posicion x del generador
                transform.position = new Vector3(transform.position.x + xSeparation, transform.position.y, transform.position.z);
            }

            //Reorganizamos el listado de cartas para que no sea posible que se vuelva a repetir
            if(random != cards.Length-1-countCards){
                card = cards[cards.Length-1-countCards];
                cards[cards.Length-1-countCards] = cards[random];
                cards[random] = card;

            }
            countCards++;

        }
    }
}
```

7.9. CardState

Aquest script està associat a totes les cartes de les escenes i la seva funcionalitat principal és gestionar l'estat, les animacions i la posició de la carta.

Per tant, definim com a variables, el component *Animator* de la carta, encarregat de les animacions, el component *Transform,* encarregat de la posició i mesures de la carta, i un seguit de variables que indiquen l'estat de la carta:

```
//Objeto para controlar las animaciones de las cartas
private Animator animator;
//Objeto para detectar las posiciones de las cartas
private Transform transform;
//Para saber si la carta esta disponible para ser seleccionada
private bool canBeSelected;
//Para saber si la carta ha sido seleccionada para mostrar el valor, inicialmente falsa.
private bool selected;
//Para saber si la carta ha sido adivinada
private bool guessed;
//Para saber si una carta es dinámica (modo dinámico)
public bool dynamic;
```

En la funció *Start* inicialitzem totes les variables i afegim els observadors per tal de ser notificat quan succeixi un esdeveniment sobre la carta. La funcionalitat que canvia d'estat i anima les cartes és:

```
void NoSelectedCards(Notification notification){
    //Cuando se seleccionen las cartas, todas incluidas estas deben pasar a ser no seleccionables hasta que se resuelva la comparacion
    this.canBeSelected = false;
}

void GuessedCards(Notification notification){
    //Si las cartas han sido iguales, lo asignamos en la variable i mantenemos la animacion
    if (this.selected == true) {
        this.guessed = true;
        this.selected = false;
        this.animator.SetBool("hidingValue", false);
        this.animator.SetBool("showingValue", true);
        //Asignamos la etiqueta de carta adivinada
        this.gameObject.tag = "Guessed";
    }
}

void NoGuessedCards(Notification notification){
    //Si las cartas no son iguales, lo indicamos en la variable y cambiamos la animacion
    if (this.selected == true) {
        this.guessed = false;
        this.selected = false;
        this.animator.SetBool("showingValue", false);
        this.animator.SetBool("hidingValue", true);
        //Devolvemos la primera etiqueta a la carta
        this.gameObject.tag = "Card";
    }
}
```

```
void DynamicNoGuessedCards(Notification notification){
    //Si las cartas no son iguales, lo indicamos en la variable y iniciamos la animacion para intercambiar posiciones
    if (this.selected == true) {
        this.guessed = false;
        this.selected = false;
        this.dynamic = true;
        this.animator.SetBool("showingValue", false);
        this.animator.SetBool("hidingValue", true);
        //Devolvemos la primera etiqueta a la carta
        this.gameObject.tag = "Card";
    }
}

void DynamicCardsToMove(Notification notification){
    //Si la carta tiene estado dinamico
    if(this.dynamic == true){
        this.gameObject.tag = "Dynamic";
    }
}

void DynamicCardsToNormal(Notification notification){
    if(this.dynamic == true){
        this.dynamic = false;

    }
}

void SelectCardsAgain(Notification notification){
    //Cuando se haya resuelto la comparacion, menos las adivinadas el resto debe ser seleccionable de nuevo para seguir la partida
    if (this.guessed == false) {
        this.canBeSelected = true;
        this.gameObject.tag = "Card";
    }
}
```

Com es pot veure, el codi és bastant simple:

- ***NoSelectedCards***: Quan es crida aquesta funció indiquem que les cartes passen a ser no seleccionables, ja que el sistema haurà de realitzar les comprovacions.

- ***GuessedCards:*** Aquesta funció es crida quan el parell, o doble parella, han estat acertats, indiquem l'estat a les variables indicadores, mantenim l'animació per a que les cartes mantinguin el valor descobert fins al final de partida i cambiem l'etiqueta de l'objecte a *Guessed*, ja que així l'usuari no podrà tornar a seleccionar-les.

- ***NoGuessedCards:*** Aquesta funció es crida quan les cartes no han estat acertades. És similar a l'anterior però amb la diferència que l'estat de l'animació canvia i, per tant, les cartes es tornen a donar la volta.

- ***DynamicNoGuessedCards:*** Aquesta funció es crida només en el mode de cartes dinàmiques, i és similar a l'anterior però amb la diferència que canviem l'estat de la carta a dinàmica. Així posteriorment, aquesta canviarà de posició.

- ***DynamicCardsToMove:*** Aquesta funció es crida amb l'intenció que, un cop assignades les cartes dinàmiques, canviem l'etiqueta d'aquestes perquè l'script encarregat de gestionar el canvi de posició sàpiga quines són les cartes dinàmiques.

- ***DynamicCardsToNormal:*** Aquesta funció es crida quan les cartes dinàmiques ja han intercanviat les seves posicions. Per tant, canviem l'estat de la carta.

- ***SelectCardsAgain:*** Aquesta funció es crida un cop realitzades totes les operacions del sistema sobre les cartes, per retornar l'estat a seleccionable a totes les cartes que no han sigut endevinades.

En la funció prncipal, que detecta que l'usuari selecciona la carta, especifiquem que, si la carta és seleccionable, reprodueixi el so corresponent, canviï l'estat de la carta, faci l'animació i envia la notificació a un altre script conforme una carta ha sigut seleccionada.

```
//Cuando el usuario toca la carta
void OnMouseDown(){
    //Si la carta no se ha adivinado ni seleccionado
    if (this.guessed == false && this.canBeSelected == true) {
        //Reproducimos el sonido de la carta
        GetComponent<AudioSource>().Play();
        //Cambiamos la etiqueta al gameobject para posteriormente detectar cartas seleccionadas
        gameObject.tag = "SelectedCard";
        //Indicamos que la carta ha sido seleccionada y cambiamos el estado en el animador para que empiece la animacion correspondiente
        this.selected = true;
        this.canBeSelected = false;
        this.animator.SetBool("hidingValue", false);
        this.animator.SetBool("showingValue", true);
        //Notificamos al centro de notificaciones que la carta ha sido seleccionada
        NotificationCenter.DefaultCenter ().PostNotification (this, "CardsSelected");
    }

}
```

Aquí es pot veure el codi complet de l'arxiu:

```
using UnityEngine;
using System.Collections;

public class CardState : MonoBehaviour {

    //Objeto para controlar las animaciones de las cartas
    private Animator animator;
    //Objeto para detectar las posiciones de las cartas
    private Transform transform;
    //Para saber si la carta esta disponible para ser seleccionada
    private bool canBeSelected;
    //Para saber si la carta ha sido seleccionada para mostrar el valor, inicialmente falsa.
    private bool selected;
    //Para saber si la carta ha sido adivinada
    private bool guessed;
    //Para saber si una carta es dinámica (modo dinámico)
    public bool dynamic;


    // Use this for initialization
    void Start () {
        //Iniciamos las variables
        this.canBeSelected = true;
        this.selected = false;
        this.guessed = false;
        //this.dynamic = false;
        this.animator = GetComponent<Animator> ();
        this.transform = GetComponent<Transform> ();
        //Añadimos un observador a este componente para que observe si la partida ha empezado.
        //NotificationCenter.DefaultCenter().AddObserver(this, "GameStart");
        //Añadimos un observador a este componente para que observe cuando las cartas no seleccionadas no se puedan seleccionar.
        NotificationCenter.DefaultCenter().AddObserver(this, "NoSelectedCards");
        //Añadimos un observador a este componente para que observe cuando las cartas son adivinadas.
        NotificationCenter.DefaultCenter().AddObserver(this, "GuessedCards");
        //Añadimos un observador a este componente para que observe cuando las cartas no son adivinadas del modo dinamico.
        NotificationCenter.DefaultCenter().AddObserver(this, "DynamicNoGuessedCards");
        //Añadimos un observador a este componente para que observe cuando las cartas dinamicas deban cambiar de etiqueta para ser animadas.
        NotificationCenter.DefaultCenter().AddObserver(this, "DynamicCardsToMove");
```

```
    //Añadimos un observador a este componente para que observe cuando las cartas dinamicas deban pasar a ser no dinamicas.
    NotificationCenter.DefaultCenter().AddObserver(this, "DynamicCardsToNormal");
    //Añadimos un observador a este componente para que observe cuando las cartas no son adivinadas.
    NotificationCenter.DefaultCenter().AddObserver(this, "NoGuessedCards");
    //Añadimos un observador a este componente para que observe cuando las cartas pueden volver a ser seleccionables.
    NotificationCenter.DefaultCenter().AddObserver(this, "SelectCardsAgain");
}

// Update is called once per frame
void Update () {

}


void NoSelectedCards(Notification notification){
    //Cuando se seleccionen las cartas, todas incluidas estas deben pasar a ser no seleccionables hasta que se resuelva la comparacion
    this.canBeSelected = false;
}

void GuessedCards(Notification notification){
    //Si las cartas han sido iguales, lo asignamos en la variable i mantenemos la animacion
    if (this.selected == true) {
        this.guessed = true;
        this.selected = false;
        this.animator.SetBool("hidingValue", false);
        this.animator.SetBool("showingValue", true);
        //Asignamos la etiqueta de carta adivinada
        this.gameObject.tag = "Guessed";
    }
}

void NoGuessedCards(Notification notification){
    //Si las cartas no son iguales, lo indicamos en la variable y cambiamos la animacion
    if (this.selected == true) {
        this.guessed = false;
        this.selected = false;
        this.animator.SetBool("showingValue", false);
        this.animator.SetBool("hidingValue", true);
        //Devolvemos la primera etiqueta a la carta
        this.gameObject.tag = "Card";
    }
}
```

```
void DynamicNoGuessedCards(Notification notification){
    //Si las cartas no son iguales, lo indicamos en la variable y iniciamos la animacion para intercambiar posiciones
    if (this.selected == true) {
        this.guessed = false;
        this.selected = false;
        this.dynamic = true;
        this.animator.SetBool("showingValue", false);
        this.animator.SetBool("hidingValue", true);
        //Devolvemos la primera etiqueta a la carta
        this.gameObject.tag = "Card";
    }
}

void DynamicCardsToMove(Notification notification){
    //Si la carta tiene estado dinamico
    if(this.dynamic == true){
        this.gameObject.tag = "Dynamic";
    }
}

void DynamicCardsToNormal(Notification notification){
    if(this.dynamic == true){
        this.dynamic = false;

    }
}

void SelectCardsAgain(Notification notification){
    //Cuando se haya resuelto la comparacion, menos las adivinadas el resto debe ser seleccionable de nuevo para seguir la partida
    if (this.guessed == false) {
        this.canBeSelected = true;
        this.gameObject.tag = "Card";
    }
}
```

```
    //Cuando el usuario toca la carta
    void OnMouseDown(){
        //Si la carta no se ha adivinado ni seleccionado
        if (this.guessed == false && this.canBeSelected == true) {
            //Reproducimos el sonido de la carta
            GetComponent<AudioSource>().Play();
            //Cambiamos la etiqueta al gameobject para posteriormente detectar cartas seleccionadas
            gameObject.tag = "SelectedCard";
            //Indicamos que la carta ha sido seleccionada y cambiamos el estado en el animador para que empiece la animacion correspondiente
            this.selected = true;
            this.canBeSelected = false;
            this.animator.SetBool("hidingValue", false);
            this.animator.SetBool("showingValue", true);
            //Notificamos al centro de notificaciones que la carta ha sido seleccionada
            NotificationCenter.DefaultCenter ().PostNotification (this, "CardsSelected");
        }


    }



}
```

7.10. ExitApplication

Aquest script està únicament associat al botó del menú *Exit* i l'única funcionalitat és tancar l'aplicació quan l'usuari toca l'opció. Per tant, el codi complet és:

```
using UnityEngine;
using System.Collections;

public class ExitApplication : MonoBehaviour {


    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }

    //Cuando el usuario toca el boton
    void OnMouseDown(){
        //Reproducimos el sonido del boton
        GetComponent<AudioSource>().Play();
        //Cerramos la aplicacion
        Application.Quit ();
    }
}
```

7.11. GameComunication

Aquest script està associat a l'objecte *GameComunication* i la funcionalitat principal és gestionar els rècords del jugador, les bases de dades i la comunicació amb les plataformes externes *Google Play* i *AdMob*.

Les variables necessàries per desenvolupar aquest fitxer han estat:

```
//Guarda la puntuacion maxima lograda por el jugador
public int maxScore = 0;
//Guarda el tiempo que ha tardado para completar el nivel
public int minTime = 0;
//Objeto de la propia clase
public static GameComunication gameCom;
//Texto que guarda la ruta del archivo donde se guardan las puntuaciones
private string fileDir;
```

- **Puntuació màxima:** És la puntuació màxima aconseguida per l'usuari.
- **Temps mínim:** Desa el temps mínim en completar la partida.
- Un objecte static propi de la classe, que perdurarà en totes les escenes.
- Una cadena de text que definirà la base de dades segons el mode de joc triat. En aquest cas, les bases de dades són arxius en format *.dat* que emmagatzemen els rècords del jugador.

En el constructor, per a que l'objecte perduri durant la comunicació entre escenes, hem d'indicar que si no existeix en la següent escena, que no es destrueixi. A més a més, cal activar la plataforma de *Google Play Games*.

```
void Awake(){
    //El objeto ha de perdurar durante el cambio de escenas
    if (gameCom == null) {
        gameCom = this;
        DontDestroyOnLoad (gameObject);
        PlayGamesPlatform.DebugLogEnabled = false;
        PlayGamesPlatform.Activate();
    } else if (gameCom != null) {
        Destroy(gameObject);
    }
}
```

Quan l'objecte es crea en l'escena, afegim els observadors corresponents a l'inici dels modes i assignem un dispositiu de prova a la plataforma AdMob, que serà el dispositiu on provem la publicitat. Cal realitzar aquest pas per no generar falses

visites a la publicitat i que el sistema ens tanqui el compte. Finalment carreguem l'anunci de tipus *Insterstícial,* amb l'id corresponent, creat anteriorment a la plataforma externa. A més a més, indiquem que si l'usuari ja havia ingressat amb el compte a Google Play Games anteriorment, no cal que torni a ingressar.

```
// Use this for initialization
void Start () {
    //Notificaciones para saber que modo de juego se esta ejecutando para cargar su respectiva base de datos
    NotificationCenter.DefaultCenter().AddObserver(this, "NormalModeSceneBegin");
    NotificationCenter.DefaultCenter().AddObserver(this, "DoublePairModeSceneBegin");
    NotificationCenter.DefaultCenter().AddObserver(this, "SoundModeSceneBegin");
    NotificationCenter.DefaultCenter().AddObserver(this, "DynamicModeSceneBegin");

    //Asignamos un id de prueba de dispositivo y cargamos el anuncio que posteriormente se mostrara
    string[] testDeviceIDs = new string[]{"7D09936E583F0CB9476601F4E6039B39"};
    EasyGoogleMobileAds.GetInterstitialManager().SetTestDevices(true, testDeviceIDs);
    EasyGoogleMobileAds.GetInterstitialManager().PrepareInterstitial("ca-app-pub-1200012394295028/9271414591");

    //Si el usuario, al iniciar el juego ya habia iniciado sesion anteriomente, se iniciara automaticamente
    ((PlayGamesPlatform)Social.Active).Authenticate ((bool success) => {}, true);

}
```

Segons el mode seleccionat, indiquem quina és la ruta de la base de dades.

```
//Segun el modo de juego, cargamos un directorio o otro
void NormalModeSceneBegin(){
    //Cargamos la ruta del archivo
    this.fileDir = Application.persistentDataPath + "/normalMode.dat";
    Load ();
}

void DoublePairModeSceneBegin(){
    //Cargamos la ruta del archivo
    this.fileDir = Application.persistentDataPath + "/doublePairMode.dat";
    Load ();
}

void SoundModeSceneBegin(){
    //Cargamos la ruta del archivo
    this.fileDir = Application.persistentDataPath + "/soundMode.dat";
    Load ();
}

void DynamicModeSceneBegin(){
    //Cargamos la ruta del archivo
    this.fileDir = Application.persistentDataPath + "/dynamicMode.dat";
    Load ();
}
```

Finalment, tenim les dos funcions principals que ens permeten guardar o carregar la puntuació màxima aconseguida pel jugador. Això ho aconseguim comunicant-nos amb la base de dades, passant la informació o recuperant-la.

```
public void Save(){
    //Creamos el archivo para guardar la puntuacion
    BinaryFormatter bf = new BinaryFormatter ();
    FileStream file = File.Create (this.fileDir);
    //Actualizamos y guardamos la puntuacion
    DataToSave data = new DataToSave ();
    data.maxScore = this.maxScore;
    data.minTime = this.minTime;

    bf.Serialize (file, data);
    file.Close ();

}

void Load(){
    //Si ya existe el archivo creado
    if(File.Exists(this.fileDir)){
        BinaryFormatter bf = new BinaryFormatter();
        FileStream file = File.Open(this.fileDir, FileMode.Open);

        DataToSave data = (DataToSave) bf.Deserialize(file);

        this.maxScore = data.maxScore;
        this.minTime = data.minTime;

        file.Close();
    }else{
        this.maxScore = 0;
        this.minTime = 0;
    }
}
```

Aquí es pot veure el codi complet de l'arxiu:

```
using UnityEngine;
using System.Collections;
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;
using System;
using GooglePlayGames;
using UnityEngine.SocialPlatforms;

public class GameComunication : MonoBehaviour {

    //Guarda la puntuacion maxima lograda por el jugador
    public int maxScore = 0;
    //Guarda el tiempo que ha tardado para completar el nivel
    public int minTime = 0;
    //Objeto de la propia clase
    public static GameComunication gameCom;
    //Texto que guarda la ruta del archivo donde se guardan las puntuaciones
    private string fileDir;

    void Awake(){
        //El objeto ha de perdurar durante el cambio de escenas
        if (gameCom == null) {
            gameCom = this;
            DontDestroyOnLoad (gameObject);
            PlayGamesPlatform.DebugLogEnabled = false;
            PlayGamesPlatform.Activate();
        } else if (gameCom != null) {
            Destroy(gameObject);
        }
    }


    // Use this for initialization
    void Start () {
        //Notificaciones para saber que modo de juego se esta ejecutando para cargar su respectiva base de datos
        NotificationCenter.DefaultCenter().AddObserver(this, "NormalModeSceneBegin");
        NotificationCenter.DefaultCenter().AddObserver(this, "DoublePairModeSceneBegin");
        NotificationCenter.DefaultCenter().AddObserver(this, "SoundModeSceneBegin");
        NotificationCenter.DefaultCenter().AddObserver(this, "DynamicModeSceneBegin");
```

```
        //Asignamos un id de prueba de dispositivo y cargamos el anuncio que posteriormente se mostrara
        string[] testDeviceIDs = new string[]{"7D09936E583F0CB9476601F4E6039B39"};
        EasyGoogleMobileAds.GetInterstitialManager().SetTestDevices(true, testDeviceIDs);
        EasyGoogleMobileAds.GetInterstitialManager().PrepareInterstitial("ca-app-pub-1200012394295028/9271414591");

        //Si el usuario, al iniciar el juego ya habia iniciado sesion anteriomente, se iniciara automaticamente
        ((PlayGamesPlatform)Social.Active).Authenticate ((bool success) => {}, true);

    }
    //Segun el modo de juego, cargamos un directorio o otro
    void NormalModeSceneBegin(){
        //Cargamos la ruta del archivo
        this.fileDir = Application.persistentDataPath + "/normalMode.dat";
        Load ();
    }

    void DoublePairModeSceneBegin(){
        //Cargamos la ruta del archivo
        this.fileDir = Application.persistentDataPath + "/doublePairMode.dat";
        Load ();
    }

    void SoundModeSceneBegin(){
        //Cargamos la ruta del archivo
        this.fileDir = Application.persistentDataPath + "/soundMode.dat";
        Load ();
    }

    void DynamicModeSceneBegin(){
        //Cargamos la ruta del archivo
        this.fileDir = Application.persistentDataPath + "/dynamicMode.dat";
        Load ();
    }

    // Update is called once per frame
    void Update () {

    }
```

```
    public void Save(){
        //Creamos el archivo para guardar la puntuacion
        BinaryFormatter bf = new BinaryFormatter ();
        FileStream file = File.Create (this.fileDir);
        //Actualizamos y guardamos la puntuacion
        DataToSave data = new DataToSave ();
        data.maxScore = this.maxScore;
        data.minTime = this.minTime;

        bf.Serialize (file, data);
        file.Close ();
    }

    void Load(){
        //Si ya existe el archivo creado
        if(File.Exists(this.fileDir)){
            BinaryFormatter bf = new BinaryFormatter();
            FileStream file = File.Open(this.fileDir, FileMode.Open);

            DataToSave data = (DataToSave) bf.Deserialize(file);

            this.maxScore = data.maxScore;
            this.minTime = data.minTime;

            file.Close();
        }else{
            this.maxScore = 0;
            this.minTime = 0;
        }
    }

    public string getTimeString(){
        return Time.timeObj.CalculateTimeToString(minTime);
    }

    [Serializable]
    class DataToSave{
        public int maxScore;
        public int minTime;
    }
}
```

7.12. GameState

Aquest script està associat a la càmera principal de cada mode de joc i és l'encarregada de gestionar la partida.

Les variables definides són les següents:

```
//Nombre de la escena a mostrar
public string nameScene;
//Booleano para saber si se han colocado las cartas
private bool cardsInGame;
//Booleano para saber si la partida ha empezado
private bool gameStarted;
//Numero de cartas seleccionadas seleccionadas y el maximo permitido
private int selectedCards;
private int maxNumSelectedCards;
//Tiempo reglamentario en segundos para la comprovacion de las cartas
private int timeToCompare;
//Integer que cuenta los aciertos seguidos del jugador
private int combo;
//Numero de cartas dinamicas y máximo permitido
private int dynamicCards;
private int maxNumDynamicCards;
```

- **El nom de l'escena:** Desem el nom de l'escena carregada, ja que segons el mode de joc tenim unes regles o unes altres.
- **Cartes en joc:** Indica si les cartes ja han estat col·locades en la pantalla.
- **Joc començat**: Indica si la partida ja ha començat.
- **Cartes seleccionades:** Indica el nombre de cartes seleccionades actualment.
- **Màxim número de cartes seleccionades:** Indica el nombre màxim de cartes que es poden seleccionar en la partida.
- **Temps per comparar:** És un valor, en segons, que indiquem per realitzar les animacions i les comprovacions del sistema.
- **Multiplicador de puntuació:** Indica el nombre d'encerts seguits que l'usuari ha aconseguit.
- **Cartes dinàmiques:** Indica el nombre actual de cartes dinàmiques hi han en la pantalla.
- **Màxima nombre de cartes dinàmiques:** Indica el nombre màxim de cartes dinàmiques possibles en la partida.

Quan s'inicia l'objecte, inicialitzem les variables segons el mode de joc triat per l'usuari.

Quan l'usuari selecciona cartes, es crida la funció que s'encarrega de notificar que, un cop s'hagin seleccionat el màxim de cartes possibles, les cartes passin a ser no

seleccionables i crida al mètode que les compara:

```
//Cuando se seleccionan las cartas
void CardsSelected(Notification notification){

    //Incrementamos el numero de cartas seleccionadas
    this.selectedCards += 1;
    //Comprovamos si ha seleccionado el numero maximo de cartas
    if (this.selectedCards == this.maxNumSelectedCards) {
        //Notificamos que las cartas ha de ser no seleccionables
        NotificationCenter.DefaultCenter().PostNotification (this, "NoSelectedCards");
        //Empezamos a comparar una vez haya pasado el tiempo reglamentario de comprovacion, para completar animaciones
        Invoke("comparingCards",this.timeToCompare);
    }
}
```

Ara passem al motor de la partida, la pròxima funció és la que gestiona les notificacions de la partida, juntament amb la normativa d'aquesta:

```
void comparingCards(){
    //Variable para detectar si las cartas son iguales
    bool equalCards = true;
    //Comprovamos si las cartas son iguales
    equalCards = checkEqualityCards();
    //Si son iguales las mantenemos mostrando el valor, si no les volvemos a dar la vuelta
    if(equalCards){
        //Actualizamos el combo del jugador
        this.combo += 1;
        //Actualizamos la puntuacion del jugador
        NotificationCenter.DefaultCenter().PostNotification (this, "UpdateScore", this.combo);
        //Notificamos que las cartas han sido adivinadas
        NotificationCenter.DefaultCenter().PostNotification (this, "GuessedCards");
        //Comprobamos si el jugador ha ganado la partida
        bool hasWon = CheckPlayerHasWon();
        if(hasWon == true){
            //Notificamos que el jugador ha ganado la partida
            NotificationCenter.DefaultCenter().PostNotification (this, "GameWon");
        }
        //Reiniciamos el numero de cartas seleccionadas
        this.selectedCards = 0;
    }else{
        //Reiniciamos el combo del jugador, ya que ha fallado y, por lo tanto, roto la racha
        this.combo = 0;
        NotificationCenter.DefaultCenter().PostNotification (this, "ResetCombo");

        //Si el modo de juego és dinámico
        if(this.nameScene == "DynamicModeScene"){
            //Incrementamos el número de cartas dinámicas
            this.dynamicCards += 2;
            //Notificamos que las cartas no han sido adivinadas
            NotificationCenter.DefaultCenter().PostNotification (this, "DynamicNoGuessedCards");
            if(this.dynamicCards == this.maxNumDynamicCards){
                NotificationCenter.DefaultCenter().PostNotification (this, "DynamicCardsToMove");
                //Recuperamos las cuatro primeras cartas y las animamos
                DynamicCardsAnimation();
                //Reiniciamos el numero de cartas dinamicas
                this.dynamicCards = 0;
            }
```

```
        //Si es cualquier otro
        }else{
            //Notificamos que las cartas no han sido adivinadas
            NotificationCenter.DefaultCenter().PostNotification (this, "NoGuessedCards");
        }

        //Reiniciamos el numero de cartas seleccionadas
        this.selectedCards = 0;
    }
    //Notificamos que el resto de cartas vuelven a ser seleccionables
    NotificationCenter.DefaultCenter().PostNotification (this, "SelectCardsAgain");
}
```

Primer de tot, recuperem si les cartes han estat encertades amb el mètode *checkEqualityCards*, que mostrarem posteriorment. En cas de retornar el valor cert:

- Augmentem el valor del multiplicado +1.
- Notifiquem que s'ha d'actualitzar la puntuació per pantalla.
- Notifiquem que les cartes han estat encertades.
- Comprovem si el jugador ja ha encertat totes les cartes. En cas de ser així, notifiquem que el jugador ja ha guanyat la partida.
- Retornem el valor de les cartes seleccionades a 0.

En cas de retornar valor fals:

- Retornem el valor del multiplicador a 0.
- Notifiquem que el multiplicador de puntuació ha de ser reiniciat.
- Si es tracta del mode de joc de cartes dinàmiques, cal augmentar el nombre de cartes dinàmiques existents, notificar que les cartes dinàmiques no han sigut encertades i, si el nombre de cartes dinàmiques és igual al màxim, cal notificar que aquestes han d'intercanviar posicions. Finalment, retornem el nombre de cartes dinàmiques a 0.
- En cas de no ser el mode dinàmic, notifiquem que les cartes no han estat encertades.
- Finalment, retornem el valor de les cartes seleccionades a 0 i notifiquem que les cartes passin a ser seleccionables de nou.

Aquí podem veure el codi complet de l'arxiu:

```
using UnityEngine;
using System.Collections;

public class GameState : MonoBehaviour {

    //Nombre de la escena a mostrar
    public string nameScene;
    //Booleano para saber si se han colocado las cartas
    private bool cardsInGame;
    //Booleano para saber si la partida ha empezado
    private bool gameStarted;
    //Numero de cartas seleccionadas seleccionadas y el maximo permitido
    private int selectedCards;
    private int maxNumSelectedCards;
    //Tiempo reglamentario en segundos para la comprovacion de las cartas
    private int timeToCompare;
    //Integer que cuenta los aciertos seguidos del jugador
    private int combo;
    //Numero de cartas dinamicas y máximo permitido
    private int dynamicCards;
    private int maxNumDynamicCards;

    // Use this for initialization
    void Start () {
        //Iniciamos variables
        this.cardsInGame = false;
        this.gameStarted = false;
        this.combo = 0;
        this.selectedCards = 0;
        this.dynamicCards = 0;
        this.maxNumDynamicCards = 4;
        //Dependiendo de la escena seleccionada, cambia el número de cartas a seleccionar
        if(this.nameScene == "NormalModeScene" || this.nameScene == "DynamicModeScene" || this.nameScene == "SoundModeScene"){
            this.maxNumSelectedCards = 2;
        }else if(this.nameScene == "DoublePairModeScene"){
            this.maxNumSelectedCards = 4;

        }
        this.timeToCompare = 1;
        //Añadimos un observador a este componente para que observe si ha habido cartas seleccionadas.
        NotificationCenter.DefaultCenter().AddObserver(this, "CardsSelected");
```

```
    }

    // Update is called once per frame
    void Update () {
        //Comprovamos si se han colocado las cartas, si no es asi notificamos el evento
        if (this.cardsInGame == false) {
            //Notificamos que iniciamos la escena
            NotificationCenter.DefaultCenter().PostNotification (this, nameScene+"Begin");
            //Cambiamos el estado del juego
            this.cardsInGame = true;

        }

        //Comprovamos si ha empezado la partida
        if (this.gameStarted == false) {
            //Notificamos que iniciamos la escena
            NotificationCenter.DefaultCenter().PostNotification (this, "GameStart");
            //Cambiamos el estado del juego
            this.gameStarted = true;

        }

    }

    //Cuando se seleccionan las cartas
    void CardsSelected(Notification notification){

        //Incrementamos el numero de cartas seleccionadas
        this.selectedCards += 1;
        //Comprovamos si ha seleccionado el numero maximo de cartas
        if (this.selectedCards == this.maxNumSelectedCards) {
            //Notificamos que las cartas ha de ser no seleccionables
            NotificationCenter.DefaultCenter().PostNotification (this, "NoSelectedCards");
            //Empezamos a comparar una vez haya pasado el tiempo reglamentario de comprovacion, para completar animaciones
            Invoke("comparingCards",this.timeToCompare);
        }
    }
```

```
void comparingCards(){
    //Variable para detectar si las cartas son iguales
    bool equalCards = true;
    //Comprovamos si las cartas son iguales
    equalCards = checkEqualityCards();
    //Si son iguales las mantenemos mostrando el valor, si no les volvemos a dar la vuelta
    if(equalCards){
        //Actualizamos el combo del jugador
        this.combo += 1;
        //Actualizamos la puntuacion del jugador
        NotificationCenter.DefaultCenter().PostNotification (this, "UpdateScore", this.combo);
        //Notificamos que las cartas han sido adivinadas
        NotificationCenter.DefaultCenter().PostNotification (this, "GuessedCards");
        //Comprobamos si el jugador ha ganado la partida
        bool hasWon = CheckPlayerHasWon();
        if(hasWon == true){
            //Notificamos que el jugador ha ganado la partida
            NotificationCenter.DefaultCenter().PostNotification (this, "GameWon");
        }
        //Reiniciamos el numero de cartas seleccionadas
        this.selectedCards = 0;
    }else{
        //Reiniciamos el combo del jugador, ya que ha fallado y, por lo tanto, roto la racha
        this.combo = 0;
        NotificationCenter.DefaultCenter().PostNotification (this, "ResetCombo");

        //Si el modo de juego és dinámico
        if(this.nameScene == "DynamicModeScene"){
            //Incrementamos el número de cartas dinámicas
            this.dynamicCards += 2;
            //Notificamos que las cartas no han sido adivinadas
            NotificationCenter.DefaultCenter().PostNotification (this, "DynamicNoGuessedCards");
            if(this.dynamicCards == this.maxNumDynamicCards){
                NotificationCenter.DefaultCenter().PostNotification (this, "DynamicCardsToMove");
                //Recuperamos las cuatro primeras cartas y las animamos
                DynamicCardsAnimation();
                //Reiniciamos el numero de cartas dinamicas
                this.dynamicCards = 0;
            }
```

```
            //Si es cualquier otro
            }else{
                //Notificamos que las cartas no han sido adivinadas
                NotificationCenter.DefaultCenter().PostNotification (this, "NoGuessedCards");
            }

            //Reiniciamos el numero de cartas seleccionadas
            this.selectedCards = 0;
        }
        //Notificamos que el resto de cartas vuelven a ser seleccionables
        NotificationCenter.DefaultCenter().PostNotification (this, "SelectCardsAgain");
    }



    //Comprovacion de igualdad de las cartas
    bool checkEqualityCards(){
        //Cartas seleccionadas
        GameObject[] cards;
        //Comprovamos si las cartas son iguales
        cards = GameObject.FindGameObjectsWithTag("SelectedCard");

        for(int i=0; i<cards.Length; i++){
            if(cards[i].name != cards[0].name){
                return false;
            }
        }

        return true;

    }
```

```
    //Comprueba si el jugador ha ganado la partida
    bool CheckPlayerHasWon(){
        //Cartas seleccionadas
        GameObject[] cards;
        //Comprovamos si hay cartas sin adivinar
        cards = GameObject.FindGameObjectsWithTag("Card");
        if(cards.Length == 0){
            return true;
        }

        return false;
    }


    void DynamicCardsAnimation(){
        //Cartas dinamicas
        GameObject[] cards;
        //Recuperamos las cartas dinamicas
        cards = GameObject.FindGameObjectsWithTag("Dynamic");

        Vector3 tras = new Vector3(0,0,0);
        tras = cards [0].gameObject.transform.position;
        //Animamos las cartas dinamicas
        for(int i=0; i<cards.Length; i++){
            if(i == cards.Length - 1){
                cards[i].gameObject.transform.position = tras;
            }else{
                cards[i].gameObject.transform.position = cards[i+1].gameObject.transform.position;
            }

        }
        //Notificamos para que cambien el estado de las cartas de nuevo
        NotificationCenter.DefaultCenter().PostNotification (this, "DynamicCardsToNormal");

    }
}
```

7.13. NotificationCenter

Aquest script no està associat a cap objecte de l'aplicació. De fet, l'única funcionalitat que té és la de comunicar esdeveniments entre els diferents objectes del joc. Aquí podem veure el codi complet de l'arxiu:

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using System;

//    NotificationCenter is used for handling messages between GameObjects.
//    GameObjects can register to receive specific notifications.  When another objects sends a notification of that typ
//    Observing GameObjects must register to receive notifications with the AddObserver function, and pass their selves,
//    Posting notifications is done by creating a Notification object and passing it to PostNotification.  All receiving
//    To use NotificationCenter, either create and manage a unique instance of it somewhere, or use the static Notificat

// We need a static method for objects to be able to obtain the default notification center.
// This default center is what all objects will use for most notifications.  We can of course create our own separate in
public class NotificationCenter : MonoBehaviour
{
    private static NotificationCenter defaultCenter;
    public static NotificationCenter DefaultCenter () {
        // If the defaultCenter doesn't already exist, we need to create it
        if (!defaultCenter) {
            // Because the NotificationCenter is a component, we have to create a GameObject to attach it to.
            GameObject notificationObject = new GameObject("Default Notification Center");
            // Add the NotificationCenter component, and set it as the defaultCenter
            defaultCenter = notificationObject.AddComponent<NotificationCenter>();
            DontDestroyOnLoad(defaultCenter);
        }

        return defaultCenter;
    }

    // Our hashtable containing all the notifications.  Each notification in the hash table is an ArrayList that contain
    Hashtable notifications = new Hashtable();

    // AddObserver includes a version where the observer can request to only receive notifications from a specific objec
    public void AddObserver (Component observer, String name) { AddObserver(observer, name, null); }
    public void AddObserver (Component observer, String name, object sender) {
        // If the name isn't good, then throw an error and return.
        if (name == null || name == "") { Debug.Log("Null name specified for notification in AddObserver."); return; }
        // If this specific notification doens't exist yet, then create it.
        if (!notifications.ContainsKey(name)) {
            notifications[name] = new List<Component>();
        }
```

```
    List<Component> notifyList = (List<Component>)notifications[name];

    // If the list of observers doesn't already contain the one that's registering, then add it.
    if (!notifyList.Contains(observer)) { notifyList.Add(observer); }
}

// RemoveObserver removes the observer from the notification list for the specified notification type
public void RemoveObserver (Component observer, String name) {
    List<Component> notifyList = (List<Component>)notifications[name]; //change from original

    // Assuming that this is a valid notification type, remove the observer from the list.
    // If the list of observers is now empty, then remove that notification type from the notifications hash.  This is for housekeeping purpos
    if (notifyList != null) {
        if (notifyList.Contains(observer)) { notifyList.Remove(observer); }
        if (notifyList.Count == 0) { notifications.Remove(name); }
    }
}

// PostNotification sends a notification object to all objects that have requested to receive this type of notification.
// A notification can either be posted with a notification object or by just sending the individual components.
public void PostNotification (Component aSender, String aName) { PostNotification(aSender, aName, null); }
public void PostNotification (Component aSender, String aName, object aData) { PostNotification(new Notification(aSender, aName, aData)); }
public void PostNotification (Notification aNotification) {
    // First make sure that the name of the notification is valid.
    if (aNotification.name == null || aNotification.name == "") { Debug.Log("Null name sent to PostNotification."); return; }
    // Obtain the notification list, and make sure that it is valid as well
    List<Component> notifyList = (List<Component>)notifications[aNotification.name]; //change from original
    if (notifyList == null) { Debug.Log("Notify list not found in PostNotification."); return; }

    // Clone list, so there won't be an issue if an observer is added or removed while notifications are being sent
    notifyList = new List<Component>(notifyList);

    // Create an array to keep track of invalid observers that we need to remove
    List<Component> observersToRemove = new List<Component>(); //change from original
```

```
        // Itterate through all the objects that have signed up to be notified by this type of notification.
        foreach (Component observer in notifyList) {
            // If the observer isn't valid, then keep track of it so we can remove it later.
            // We can't remove it right now, or it will mess the for loop up.
            if (!observer) { observersToRemove.Add(observer);
            } else {
                // If the observer is valid, then send it the notification.  The message that's sent is the name of the notification.
                observer.SendMessage(aNotification.name, aNotification, SendMessageOptions.DontRequireReceiver);
            }
        }

        // Remove all the invalid observers
        foreach (Component observer in observersToRemove) {
            notifyList.Remove(observer);
        }
    }
}

// The Notification class is the object that is sent to receiving objects of a notification type.
// This class contains the sending GameObject, the name of the notification, and optionally a hashtable containing data.
public class Notification {
    public Component sender;
    public String name;
    public object data;

    public Notification (Component aSender, String aName) { sender = aSender; name = aName; data = null; }
    public Notification (Component aSender, String aName, object aData) { sender = aSender; name = aName; data = aData; }
}
```

7.14. RankingButton

Aquest script està associat únicament al botó del menú corresponent a la classificació i la responsabilitat d'aquest arxiu és:

- Si l'usuari ha ingresat prèviament a la plataforma Google Play Games, el botó es mostrarà de color verd, sinò es mostrarà gris.

- Quan l'usuari el toca, si és de color gris, la plataforma externa demanarà a l'usuari per ingresar amb el compte. En cas contrari, la plataforma externa mostrarà la classificació actual.

Aquí es pot veure el codi complet de l'arxiu:

```
using UnityEngine;
using System.Collections;
using GooglePlayGames;
using UnityEngine.SocialPlatforms;

public class RankingButton : MonoBehaviour {

    private TextMesh textButton;

    void Awake(){
        textButton = GetComponent<TextMesh>();
    }

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {
        textButton.color = Social.localUser.authenticated ? Color.green : Color.gray;
    }

    void OnMouseDown(){
        GetComponent<AudioSource>().Play();

        if(Social.localUser.authenticated){
            ((PlayGamesPlatform)Social.Active).ShowLeaderboardUI("CgkIwtSQmsYEEAIQBg");
        }else{
            Social.localUser.Authenticate((bool success) => {});
        }
    }
}
```

7.15. Score

Aquest script està associat a l'objecte corresponent al marcador de puntuació de qualsevol mode de joc. La funcionalitat principal d'aquest arxiu és gestionar el sistema de puntuació del joc.

Primerament, com a variables necessitem:

- El component *TextMesh* de l'objecte del marcador, per anar actualitzant el marcador per pantalla a mesura que l'usuari guanyi punts.
- La puntuació bàsica que guanya l'usuari al encertar un parell o grup de cartes. En aquest joc, està definida a 50 punts.
- La puntuació actual que té l'usuari.
- El multiplicador de puntuació actual aconseguit per l'usuari.
- Un indicador que ens informa si és el primer encert de la partida.

```
//El componente que muestra la puntuación
public TextMesh scoreText;
//Puntuacion basica para el juego
private int mainScore;
//Integer que almacena la puntuación actual
private int score;
//Integer que almacena el combo que realiza el jugador
private int combo;
//Booleano para saber si es el primer acierto del jugador
private bool firstGuessedCards;
```

A l'iniciar l'objecte definim els valors de les variables i afegim els observadors necessaris per escoltar els esdeveniments. A més a més, en la funció *update*, actualitzem el marcador per pantalla. Si el jugador té un multiplicador més gran que 1, també es mostra per pantalla:

```
// Update is called once per frame
void Update () {
    //Si el jugador mantiene un combo se muestra el multiplicador al lado
    if (this.combo > 1) {
        //Mostramos por pantalla la puntuación actual
        this.scoreText.text = this.score.ToString () + " x" + this.combo.ToString ();
    } else {
        //Mostramos por pantalla la puntuación actual
        this.scoreText.text = this.score.ToString();
    }

}
```

La funció updateScore és l'encarregada d'actualitzar la puntuació segons l'estat de la partida. Si és el primer encert de l'usuari, la puntuació pasa a ser la puntuació bàsica del joc. En canvi, si no és així, es recupera el multiplicador actual i s'aplica a l'operació següent, de manera que:

```
void UpdateScore(Notification notification){
    //Si es la primera vez que acierta, no tiene puntuación y por lo tanto no se le puede multiplicar combo
    //Se asigna la puntuación base al primer acierto.
    if (this.firstGuessedCards == false){
        this.score = this.mainScore;
        this.firstGuessedCards = true;
    }else{
        //Recogemos el combo logrado por el jugador de la notificacion y actualizamos la puntuación actual
        this.combo = (int) notification.data;
        this.score += this.mainScore * combo;

    }


}
```

Finalment, l'altre mètode principal d'aquest arxiu gestiona els assoliments i envia la puntuació i temps al *GameComunication* esmentat anteriorment:

- Si el jugador aconsegueix una puntuació major que la del seu rècord,

actualitzem les dades del *GameComunication* i les desem a la base de dades.

- Si el jugador empata en punts amb el seu propi rècord, comparem els temps. Si ha trigat menys temps que en el rècord, llavors s'actualitzen les dades i es desen a la base de dades.

- Seguidament, enviem la puntuació aconseguida a la classificació de la plataforma *Google Play Games.*

- També, comprovem si l'usuari ha aconseguit un assoliment nou. En cas de ser així, l'activem.

- Finalment, enviem una notificació perquè es mostrin els resultats de la partida.

```
void GameWon(){
    //Si el jugador supera su récord, actualizamos la puntuación y el tiempo que ha tardado en conseguirlo.
    if (this.score > GameComunication.gameCom.maxScore) {
        GameComunication.gameCom.maxScore = this.score;
        GameComunication.gameCom.minTime = Time.timeObj.getTimeGame();
        GameComunication.gameCom.Save();
    //En caso de obtener la misma puntuación, miramos que el tiempo sea menor
    }else if (this.score == GameComunication.gameCom.maxScore) {
        if(Time.timeObj.getTimeGame() < GameComunication.gameCom.minTime){
            GameComunication.gameCom.maxScore = this.score;
            GameComunication.gameCom.minTime = Time.timeObj.getTimeGame();
            GameComunication.gameCom.Save();
        }

    }

    //Enviamos al ranking de google play la puntuacion lograda
    Social.ReportScore (this.score,"CgkIwtSQmsYEEAIQBg", (bool success) => {});
    //Tambien comprobamos si el jugador puede desbloquear algun logro con la puntuacion obtenida
    if (this.score >= 850) Social.ReportProgress ("CgkIwtSQmsYEEAIQAQ", 100.0, (bool success) => {});
    if (this.score >= 1200) Social.ReportProgress ("CgkIwtSQmsYEEAIQAw", 100.0, (bool success) => {});
    if (this.score >= 4500) Social.ReportProgress ("CgkIwtSQmsYEEAIQBA", 100.0, (bool success) => {});
    if (this.score >= 6000) Social.ReportProgress ("CgkIwtSQmsYEEAIQBQ", 100.0, (bool success) => {});
    if (this.score >= 6800) Social.ReportProgress ("CgkIwtSQmsYEEAIQAg", 100.0, (bool success) => {});



    //Finalmente, notificamos a la funcion que activa la victoria, mostrando los resultados actualizados de la victoria
    NotificationCenter.DefaultCenter().PostNotification (this, "GameResults");


}
```

Aquí es pot veure el codi complet de l'arxiu:

```
using UnityEngine;
using System.Collections;
using GooglePlayGames;
using UnityEngine.SocialPlatforms;

public class Score : MonoBehaviour {

    //El componente que muestra la puntuación
    public TextMesh scoreText;
    //Puntuacion basica para el juego
    private int mainScore;
    //Integer que almacena la puntuación actual
    private int score;
    //Integer que almacena el combo que realiza el jugador
    private int combo;
    //Booleano para saber si es el primer acierto del jugador
    private bool firstGuessedCards;


    // Use this for initialization
    void Start () {
        //Iniciamos variables
        this.score = 0;
        this.combo = 0;
        this.firstGuessedCards = false;
        this.mainScore = 50;
        //Añadimos un observador a este componente para que observe si hay que sumar puntuación.
        NotificationCenter.DefaultCenter().AddObserver(this, "UpdateScore");
        //Añadimos un observador a este componente para que observe si hay que resetear el combo.
        NotificationCenter.DefaultCenter().AddObserver(this, "ResetCombo");
        //Añadimos un observador a este componente para que observe si el jugador ha ganado
        NotificationCenter.DefaultCenter().AddObserver(this, "GameWon");

    }
```

```
// Update is called once per frame
void Update () {
    //Si el jugador mantiene un combo se muestra el multiplicador al lado
    if (this.combo > 1) {
        //Mostramos por pantalla la puntuación actual
        this.scoreText.text = this.score.ToString () + " x" + this.combo.ToString ();
    } else {
        //Mostramos por pantalla la puntuación actual
        this.scoreText.text = this.score.ToString();
    }

}

void UpdateScore(Notification notification){
    //Si es la primera vez que acierta, no tiene puntuación y por lo tanto no se le puede multiplicar combo
    //Se asigna la puntuación base al primer acierto.
    if (this.firstGuessedCards == false){
        this.score = this.mainScore;
        this.firstGuessedCards = true;
    }else{
        //Recogemos el combo logrado por el jugador de la notificacion y actualizamos la puntuación actual
        this.combo = (int) notification.data;
        this.score += this.mainScore * combo;

    }


}

void ResetCombo(){
    this.combo = 0;
}
```

```
    void GameWon(){
        //Si el jugador supera su récord, actualizamos la puntuación y el tiempo que ha tardado en conseguirlo.
        if (this.score > GameComunication.gameCom.maxScore) {
            GameComunication.gameCom.maxScore = this.score;
            GameComunication.gameCom.minTime = Time.timeObj.getTimeGame();
            GameComunication.gameCom.Save();
        //En caso de obtener la misma puntuación, miramos que el tiempo sea menor
        }else if (this.score == GameComunication.gameCom.maxScore) {
            if(Time.timeObj.getTimeGame() < GameComunication.gameCom.minTime){
                GameComunication.gameCom.maxScore = this.score;
                GameComunication.gameCom.minTime = Time.timeObj.getTimeGame();
                GameComunication.gameCom.Save();
            }

        }

        //Enviamos al ranking de google play la puntuacion lograda
        Social.ReportScore (this.score,"CgkIwtSQmsYEEAIQBg", (bool success) => {});
        //Tambien comprobamos si el jugador puede desbloquear algun logro con la puntuacion obtenida
        if (this.score >= 850) Social.ReportProgress ("CgkIwtSQmsYEEAIQAQ", 100.0, (bool success) => {});
        if (this.score >= 1200) Social.ReportProgress ("CgkIwtSQmsYEEAIQAw", 100.0, (bool success) => {});
        if (this.score >= 4500) Social.ReportProgress ("CgkIwtSQmsYEEAIQBA", 100.0, (bool success) => {});
        if (this.score >= 6000) Social.ReportProgress ("CgkIwtSQmsYEEAIQBQ", 100.0, (bool success) => {});
        if (this.score >= 6800) Social.ReportProgress ("CgkIwtSQmsYEEAIQAg", 100.0, (bool success) => {});



        //Finalmente, notificamos a la funcion que activa la victoria, mostrando los resultados actualizados de la victoria
        NotificationCenter.DefaultCenter().PostNotification (this, "GameResults");


    }

    public int getScore(){
        return this.score;
    }
}
```

7.16. Time

Aquest script està associat a l'objecte corresponent al temporitzador de qualsevol mode de joc. La funcionalitat principal d'aquest arxiu és gestionar el sistema de cronometratge del joc.

Les variables que necessitem són:

```
//Booleano para ver si esta en marcha el reloj
private bool countdownStarted;
//Valor del tiempo en segundos para finalizar la partida
public int timer;
//Valor del tiempo en segundos que tarda el usuario en ganar la partida
private int timeGame;
//El componente que muestra el cronometro
public TextMesh time;
//Objeto propio de la clase
public static Time timeObj;
```

- Un indicador que informi si el cronòmetre ha començat el compte enrere.
- El temps en segons que requereix la partida per finalitzar. És a dir, assignem des de l'entorn gràfic la quantitat de temps que dura la partida en segons.
- El temps que triga l'usuari en completar la partida.

- El component *TextMesh* del temporitzador, ja que haurem d'anar actualitzan el temps.

- Un objecte de la pròpia classe, que es comunicarà amb altres objectes.

Quan la partida comença, la principal funció del temporitzador és començar el compte enrere. Per tant, cal anar disminuïnt el temps segon a segon de la manera següent:

```
void Countdown(){
    //Invocamos funcion cada segundo, para restar el tiempo (InvokeRepeating una vez empieza no se puede para hasta finalizar script o escena)
    InvokeRepeating( "DecreaseTime", 1, 1 );

}

void DecreaseTime(){
    //Reducimos en 1 el valor del tiempo hasta que llegue a 0
    if (timer > 0) {
        timer--;
        this.timeGame++;
    }
    //Una vez finalizado el tiempo reglamentario, finalizamos la partida
    if (timer == 0) {
        CancelInvoke("DecreaseTime");
        //Anunciamos la derrota del jugador por falta de tiempo
        NotificationCenter.DefaultCenter().PostNotification (this, "GameOver");
    }

}
```

En cas que el temps s'hagi acabat, es comunicarà mitjançant una notificació que l'usuari ha perdut la partida. També cal aturar el temps si l'usuari completa la partida abans de que el temps finalitzi. Finalment, actualitzem per pantalla el temps que li queda a l'usuari per completar el nivell amb la funció CalculateTimeToString, que calcula els segons i els transforma a una cadena de text formatejada com es veurà posteriorment.

```
// Update is called once per frame
void Update () {
    //Mostramos el valor actual del reloj, manteniendo la forma
    time.text = CalculateTimeToString(this.timer);
}
```

Aquí es pot veure el codi complet de l'arxiu:

```
using UnityEngine;
using System.Collections;

public class Time : MonoBehaviour {

    //Booleano para ver si esta en marcha el reloj
    private bool countdownStarted;
    //Valor del tiempo en segundos para finalizar la partida
    public int timer;
    //Valor del tiempo en segundos que tarda el usuario en ganar la partida
    private int timeGame;
    //El componente que muestra el cronometro
    public TextMesh time;
    //Objeto propio de la clase
    public static Time timeObj;


    // Use this for initialization
    void Start () {
        //Inicializamos variables
        this.countdownStarted = false;
        this.timeGame = 0;
        timeObj = this;
        //Añadimos un observador a este componente para que observe el evento llamara a esa funcion cuando se notifique.
        NotificationCenter.DefaultCenter().AddObserver(this, "GameStart");
        //Añadimos un observador para que observe cuando el jugador gane la partida
        NotificationCenter.DefaultCenter ().AddObserver (this, "GameWon");
    }

    void GameStart(Notification notification){
        //Cuando la partida haya empezado, que empiece la cuenta atras
        countdownStarted = true;
        Countdown ();
    }

    void Countdown(){
        //Invocamos funcion cada segundo, para restar el tiempo (InvokeRepeating una vez empieza no se puede para hasta finalizar script o escena)
        InvokeRepeating( "DecreaseTime", 1, 1 );

    }
```

```
    void DecreaseTime(){
        //Reducimos en 1 el valor del tiempo hasta que llegue a 0
        if (timer > 0) {
            timer--;
            this.timeGame++;
        }
        //Una vez finalizado el tiempo reglamentario, finalizamos la partida
        if (timer == 0) {
            CancelInvoke("DecreaseTime");
            //Anunciamos la derrota del jugador por falta de tiempo
            NotificationCenter.DefaultCenter().PostNotification (this, "GameOver");
        }

    }

    void GameWon(){
        //Si el jugado gana la partida, paramos el tiempo
        CancelInvoke("DecreaseTime");
    }


    // Update is called once per frame
    void Update () {
        //Mostramos el valor actual del reloj, manteniendo la forma
        time.text = CalculateTimeToString(this.timer);
    }

    public int getTimeGame(){
        return this.timeGame;
    }
}
```

```
    public string CalculateTimeToString(int secs){
        //Calculamos minutos y segundos
        int hour = (secs / 3600);
        int min = ((secs - hour * 3600) / 60);
        int sec = secs - (hour * 3600 + min * 60);

        //Mostramos el valor actual del reloj, manteniendo la forma
        if (min < 10) {
            if (sec > 9) {
                return "0" + min + ":" + sec;
            } else {
                return "0" + min + ":0" + sec;
            }
        } else {
            if (sec > 9) {
                return min + ":" + sec;
            } else {
                return min + ":0" + sec;
            }
        }
    }

}
```

7.17. UpdateScoreTable

Aquest script està associat a l'objecte tal. La funcionalitat principal d'aquest arxiu actualitzar i formatejar els resultats de la finestra de victoria. Per tant, recuperem el temps i puntuació aconseguida per l'usuari en la partida actual i la mostrem per pantalla. Aquí es pot veure el codi complet de l'arxiu:

```
using UnityEngine;
using System.Collections;

public class UpdateScoreTable : MonoBehaviour {

    public TextMesh scoreT;
    public TextMesh recordT;
    public Score score;

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }

    void OnEnable(){
        int t = Time.timeObj.getTimeGame ();
        string time = Time.timeObj.CalculateTimeToString(t);
        scoreT.text = score.getScore ().ToString ()+" IN "+time;
        //scoreT.text = score.getScore ().ToString ();
        if (GameComunication.gameCom != null) {
            recordT.text = GameComunication.gameCom.maxScore.ToString()+" IN "+GameComunication.gameCom.getTimeString();
            //recordT.text = GameComunication.gameCom.maxScore.ToString();
        }
    }
}
```

8. Webgrafia

- docs.unity3d.com/es/
- es.wikipedia.org
- youtube.com